Doctoral Thesis

Hardware Design and Star Selection Algorithm for Arcsecond Pico Star Tracker

각초 별 추적기를 위한 하드웨어 설계 및 별 선택 알고리즘

February 2018

Graduate School of Seoul National University

Department of Mechanical and Aerospace Engineering

(Major: Aerospace Engineering)

Muruganandan Vishnu Anand

Hardware Design and Star Selection Algorithm for Arcsecond Pico Star Tracker

Advisor: Prof. In-Seuck Jeung

Submitting doctoral thesis of Aerospace Engineering

December 2017

Graduate School of Seoul National University

Department of Mechanical and Aerospace Engineering

(Major: Aerospace Engineering)

Muruganandan Vishnu Anand

Confirming the doctoral thesis written by Muruganandan Vishnu Anand

December 2017

Chair Youdan Kim (Seal)

Vice Chair In-Seuck Jeung (Seal)

Examiner Chang Don Kee (Seal)

Examiner Chan Gook Park (Seal)

Examiner Hyeongjun Park (Seal

초 록

별 추적기는 선험적 지식 없이 인공위성의 자세 지향을 3축 방향에 대하여 각초 단위로 추정한다. 그런데 별 추적기는 크기가 크고 무거우며 전력 소모가 크기 때문에 나노 위성 임무에 사용하기에는 적합하지 않다. 각초 피코 별 추적기(APST)는 나노 위성의 한계를 기반으로 설계되었으며, 각초 단위로 자세 지향 정보를 제공할 것으로 추정된다. APST는 COTS 구성품을 사용하여 개발함으로써 개발 시간을 단축하고 비용을 절감하였다. APST에 사용되는 COTS 구성품 (이미지 센서, 렌즈 및 배플)의 성능을 평가하기 위해 이론적 모델을 개발하였다. 이 모델을 사용하여 COTS 구성품이 별 추적기의 요구 사항을 충족하는지 확인할 수 있다. 그러나 COTS 구성품에는 이미지 센서 노이즈, 렌즈 왜곡 및 수차 등이 존재하기 때문에 APST의 전반적인 성능을 저하시킨다. 렌즈 왜곡과 부정확한 별 위치 식별은 APST의 오차에 큰 영향을 미친다. 방사형 렌즈 왜곡은 각도 거리 측정에 오류를 유발하여 별의 위치를 오인하거나 별을 식별하지 못하고, 별을 식별하는데 소용되는 시간을 증가시킨다. 이

로 인해 APST는 제 역할을 수행하지 못하게 된다. 이를 극복하기 위해 별의 각도 거리 정보를 기반으로 별을 선택하는 상대적인 별 선택 방법 (Relative star selection method)을 개발하였다. 낮은 각도 거리를 갖는 별의 쌍이 최소 측정 오차를 갖는다는 사실에 기반하여, 상대적인 별 선택은 낮은 측정 오차를 갖는 4 개의 별을 선택한다. 이는 전통적인 방식인 별의 밝기에 따라 선택하는 방식과는 구별된다. 상대적인 선택 알고리즘은 별 시뮬레이터를 이용해 75 개 별자리에 대하여 시험되었으며 100 % 성공률을 보였고, 조준 오차는 71 각초의 정확도를 가졌다. 반면, 전통적인 방식인 밝은 별을 선택하는 방식은 28%의 성공률을 보였다. 이는 별의 각도 거리에 기반하지 않았기 때문이다. 따라서 상대적인 별 선택 방식이 APST에 더 효율적이다.

주요어 : 나노 위성, 피코 별 추적기, 별 선택 알고리즘

학 번 : 2015-30841

Abstract

Hardware Design and Star Selection Algorithm for Arcsecond Pico Star Tracker

Name: Muruganandan Vishnu Anand

Department: Department of Mechanical and Aerospace Engineering

Graduate School: Seoul National University

The star tracker estimates pointing knowledge of a satellite in arcsecond accuracy in three axes without apriori knowledge. But star trackers are larger in size, heavier, power hungry and expensive for nanosatellite missions. The Arcsecond Pico Star Tracker (APST) is designed based on the limitations of nanosatellites and estimated to provide pointing knowledge in arcsecond. The APST is developed using fully COTS components because it's affordable, and less development time. A theoretical model is developed to estimate the performance of the COTS components (image sensor, lens, and baffle) used in the APST. Using this model, it's possible to validate if the components meet the requirements of the star tracker. But COTS component decreases the overall performance due to the errors in image sensor noise, lens distortion, and aberration etc. In APST, the lens distortion and inaccurate centroiding are the dominant error sources. The radial lens distortion causes an error in angular distance measurement, which leads misidentifying or identification of stars and high processing time. This leads to functional failure of APST. To overcome this, the relative star selection method is

developed which selects the stars based on the angular distance information. Based on the fact that star pair with low angular distance has minimum measurement error, the relative star selection selects the four stars with low measurement error. It's compared with conventional bright star selection method, whereas stars are selected based on brightness. The relative selection algorithm is tested with 75-star constellation in star simulator and it has delivered 100% success rate and accuracy of 71 arcseconds in boresight. Whereas the conventional bright star selection delivered low success rate of 28% because the star pairs are not selected based on angular distance separation. Hence the relative star selection algorithm is efficient for APST.

Keywords: Nanosatellites, Pico Star Tracker, Star Selection Algorithm

Student ID.: 2015-30841

Contents

List of Figures

List of Tables

Chapter 1: Introduction

In the 20th century, the birth of artificial satellite has revolutionized our understanding of the earth and universe. The artificial satellites are the eyes which see the future of the mankind. Right from the weather forecast, navigation, and communication, artificial satellites fulfills all the needs of humans in day to day life. But still, the satellite development and usage are limited to military and national defense authorities due to its high cost and secured technology. But in the 21st century, the development of small satellites has made its technology and information accessible to students in the university platform. Compared to the conventional satellites, the small satellites are smaller in size and weight. The small satellites are classified based on mass into mini, micro, nano, pico and Femto satellites [1], which is shown in Table 1.

Table 1: Classification of small satellites [1]

Category	Mass (Kg)
Mini Satellite	100 - 180
Micro Satellite	10 - 100
Nano Satellite	1 - 10
Pico Satellite	0.01 - 1
Femto Satellite	0.001 – 0.01

The number of small satellites in space are increasing every year. The primary reason for the increase in a number of small satellites is low cost and short time for development and launching. Figure 1 shows the micro and nanosatellites launch history and forecast from 2010 to 2023. In 2010 only 30 (micro and nano) satellites are launched into orbit. Based on the market demand in 2020, it estimated that around 332 (micro and nano) satellites will be launched into orbit [2]. This shows

that private investment in small satellites is increasing every year.

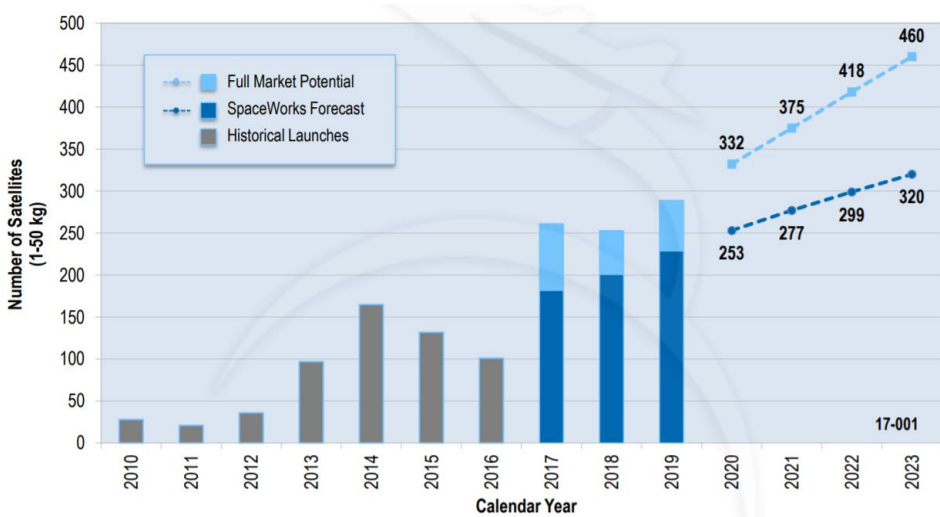

Figure 1: Micro and nanosatellites launch history and forecast [2]

In past twenty years, the small satellites have performed various science and technology demonstration mission like a constellation, remote sensing, telecommunication, astrometry. Due to less development time, low cost and range of applications, the universities all over the world are developing nano and picosatellites. The positive side of this development is students at universities obtain the firsthand experience in developing the nano and picosatellites. Even private industries Planet labs, Clyde space, ISIS in small satellites have flourished in past ten years. In short, pico and nanosatellites have made space accessible for mankind. Figure 2 shows the image of a dove (nanosatellite) from planet labs [3] and pearls from sky and space [4]. Around 150 doves are launched into orbit to image the whole world every day. This will give more detailed understanding of our earth. The sky and space will launch around 200 nano pearls for providing telecommunication in a remote part of the earth. Those two examples show the

importance and advantages of using nanosatellites in orbit.

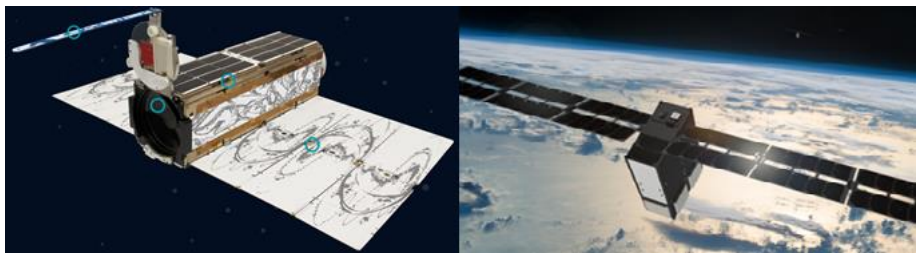

Figure 2: Dove (planet labs) and pearls (sky and space) [3] [4]

In order to perform various scientific and technology demonstration missions like formation flying, satellite constellation, telecommunication, remote sensing, astrometry and interplanetary exploration, the subsystem of nanosatellite has to upgrade. One of the subsystems needs the upgrade is attitude determination of the nanosatellite. In order to do the scientific missions, it's crucial to know the attitude of the satellite with arcsecond accuracy. But the conventional nanosatellites use sun, earth and magnetometer sensor which provides the attitude accuracy of satellite in degree.

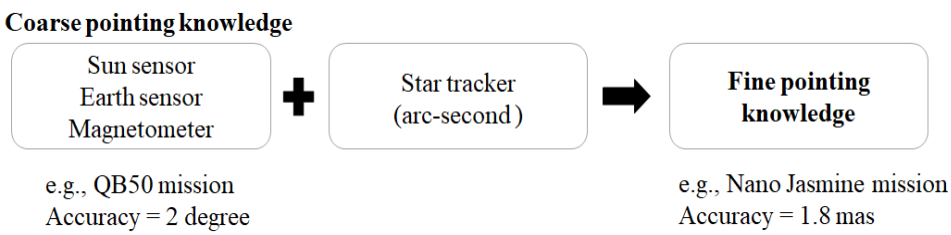

Figure 3: Attitude determination for small satellites

In order to obtain arc-second accuracy, the star tracker is required for nanosatellite. Figure 3 shows that the conventional nanosatellites (e.g., QB50) use coarse pointing knowledge which can be upgraded to fine pointing knowledge using star

tracker (e.g., Nano Jasmine) [5]. The conventional star tracker is large in size, heavy, power consuming and expensive for nanosatellite platform. Figure 4 shows the examples of conventional star tracker used in large satellites CT-632 from Ball Aerospace, Astro-APS from Jena Optronik, and HE-5AS from Terma. The properties of these star tracker are shown in Table 2. As it's clear that these star tracker cost around $ 400 k which is very expensive for nanosatellites. But it provides better accuracy of 1 to 10 arcsecond [6].

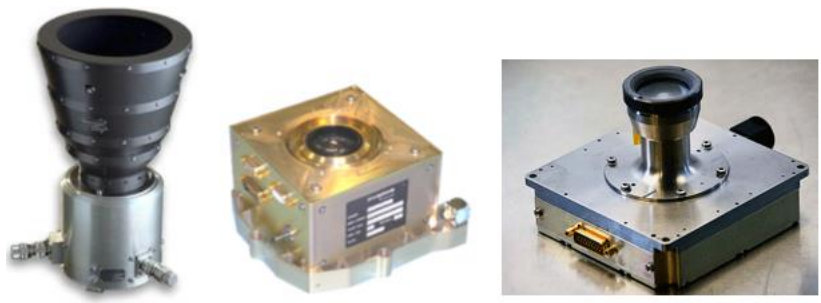

Figure 4: CT-632, Astro-APS, HE-5AS [7][8][9]

This shows clearly that there is a huge demand for star tracker which suits nanosatellite platform, it's named as pico star tracker. These pico star trackers should be less cost and fulfill all the requirements of nanosatellite. Which means it should be capable to produce accuracy in arcsecond within limited size, weight, power, and cost. Those are the challenges in developing pico star tracker.

Table 2: Properties of typical star trackers [6][9]

Star Tracker	Mass (kg)	Power (W)	Accuracy (arc-sec)	Cost ($)
CT-632	2.5	8	11	350 k
HE-5AS	1.2	5.5	1	450 k
Astro-APS	1.8	4	6	--

1.1 Background

The SNUSAT-2 (nanosatellite) is technology demonstration mission developed at Seoul National University. The primary mission of SNUSAT-2 is to monitor the health of vegetation and water sources over the predefined locations. The SNUSAT-2 contains the space camera with a resolution of 30 m and it needs accurate pointing knowledge in order to image the predefined location on earth. Figure 5 shows the image of the SNUSAT-2. The space camera images the predefined locations on the earth and monitors the health of the vegetation crops, forest and water level over a year.

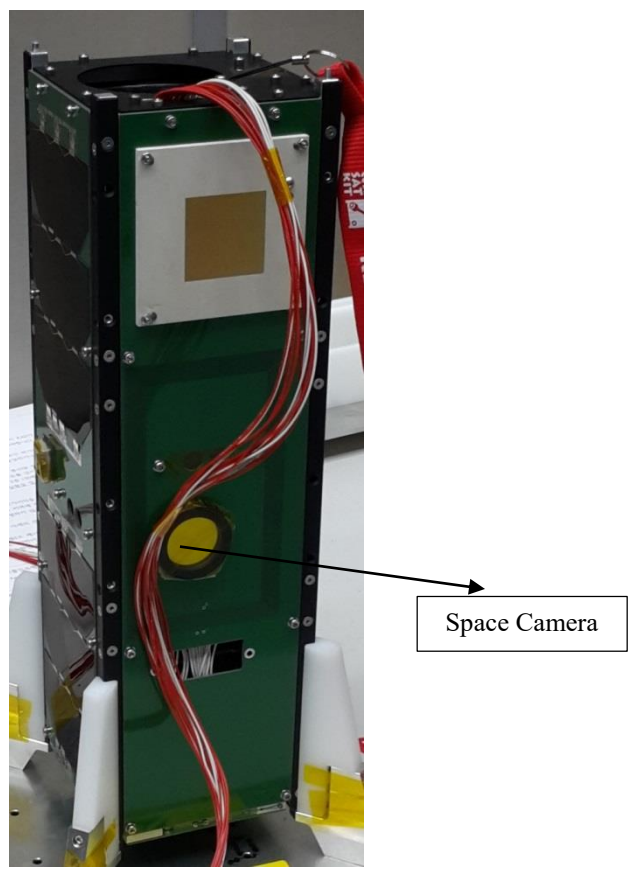

Figure 5: Seoul National University SATellite (SNUSAT-2)

The SNUSAT-2 contains sun sensor and magnetometer. Two vectors are required to determine the attitude of the satellite in three axes. But there are two disadvantages with current sensors. First, the sun sensor and magnetometer can determine attitude during the daytime but during eclipse, sun vector can't be determined. Second, both sun sensor and magnetometer determines the attitude with the accuracy in degree but SNUSAT-2 has a narrow angle of view camera of 6.5°×4.9° (Ground Swath = 68×51km). If the pointing knowledge of satellite is in degree then target location will be drifted with $\pm 1°$ error. In order to overcome this, we need pointing knowledge of the satellite with arcsecond accuracy.

Considering the above two drawbacks, star tracker is the possible solution. The star tracker can produce two vectors and it determines the attitude with arcsecond accuracy in three axes without any prior attitude knowledge. Hence we decided to use sun sensor, magnetometer, and star tracker in order to fulfill SNUSAT-2 mission. Figure 6 shows the image of sun sensor and magnetometer used in SNUSAT-2 nanosatellite. The star tracker should fit within the size, weight, and power requirement of SNUSAT-2. There are two options either to purchase the pico star tracker from the commercial market or to develop the pico star tracker completely in-house.

Figure 6: Attitude Sensors in SNUSAT-2

The star trackers for pico and nanosatellites are developed by various institutions around the world. In general, pico star trackers weigh up to 90 g but this does not include the weight of the baffle. The STC-2 has the lowest weight of 65 g, developed by Sternberg Astronomical Institute, Lomonosov Moscow State University [10]. The size of the pico star tracker is designed to fit within 0.5 U (50 x 100 x 100mm) which is half the size of a CubeSat platform. The ST-200 is the smallest pico star tracker at 30 x 30 x 38.1 mm^3 without including the baffle developed by Berlin Space Technologies, Germany. The ST-200 has the lowest nominal power consumption of 220 mW [11]. The ST-16 has the highest accuracy of 7 arcseconds (pitch/yaw) and 70 arcseconds (roll) developed by Ryerson University, Canada [12].

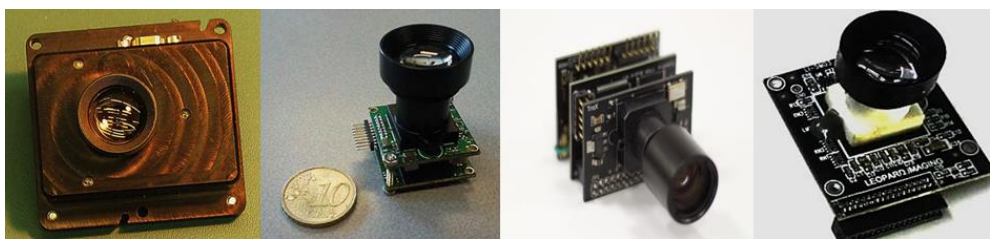

Figure 7: ST-16, ST-200, CubeStar, Picostellar gyroscope

When lost in space (LIS) apriori information about satellite is unknown; hence, the update rate is lower. However, when in tracking mode (TM) apriori information about satellite is known using other sensors; hence, there is a higher update rate. The nominal required slew rate in LEO is 0.1° to 0.3°/s. When conducting a satellite maneuver higher than the nominal slew rate, the image of a star will blur. But ST-16 is operational up to 3°/s by post-processing the image. Most pico star trackers use a low-resolution image sensor having 1 megapixel (MP)

resolution or less. However, the ST 16 and ST 200 have high-resolution image sensor of 4 MP and 5MP, respectively. The selection of image sensor includes various aspects, which will be detailed in the next sections. The important parameters of current pico star trackers are shown in Table. 2. [2]

Table 3: Properties of Current Pico Star Trackers

Name	Weight (g)	Power (mW)	Accuracy (arc-sec)	FOV (deg)	Limiting the	Update rate (Hz)
ST-16	90	250	12	20	5.75	1
ST-200	74	220	30	20	6	1
Cube star	90	350	36	42	3.8	1
STC-2	65	250	10	20	5.5	--
Pico star	70	250	36	12	6.5	4

Due to the limitation in size the pico star tracker uses imaging lenses with a low focal number, which comparably produces brighter images. The focal ratio of 1.2 to 1.8 is optimum for pico star tracker. Most pico star trackers are designed with a FOV of less than 20 but the CubeStar developed by the University of Stellenbosch, South Africa uses wide FOV (WFOV) of 42˚ that requires fewer stars to be cataloged but has relatively low accuracy and is vulnerable to stray light [13]. Based on the FOV requirement, the focal length of the imaging lens varies from 6 to 16 mm. The focal length of 16 mm would be the upper limit due to the constraints of weight, size, and accuracy. The limiting magnitude of the star tracker is the magnitude of the faintest star detectable by the star tracker. The pico star tracker developed by the University of Wuerzburg, Germany has a high sensitivity image sensor; hence, it can detect stars up to a magnitude of 6.46[14]. The ST-16 pico star tracker was launched into space as part of SkySat-1 in 2013; however, it performed lower than the expectations due to chromatic aberration of

the lens. The ST-16 RT, a modified version equipped with a customized lens, was launched as part of skySat-2 and skySat-2C in 2014 and 2016 respectively [15].

Table 4: Properties of image sensor and lens used in current pico star trackers

Name	Resolution (MP)	Pixel Size (µm)	Sensitivity (V/lux.s)	Focal ratio	Focal length (mm)
ST-16	5	2.2	1.2	1.2	16
ST-200	4	2.2	1.4	--	--
Cube star	0.5	5.6	--	1.2	6
STC-2	0.6	10	--	1.17	10.5
Pico star	0.3	5.6	16.5	1.8	16

The typical star tracker algorithm consists of three important steps, image processing, identification and attitude determination which is shown in Figure 8. Image processing is containing three divisions, which are thresholding, grouping and centroiding. The thresholding extracts the signal pixels from noise background and there are two types of thresholding global and local [16]. A star spreads over a number of pixels due to slight defocusing and all these pixels are grouped into a single star using functions like labeling and clustering [17]. The pixel size and focal length limit the star tracker accuracy. To obtain sub-pixel accuracy, the star tracker is purposely defocused to spread over many pixels (i.e. PSF of the star image is enhanced). A centroid algorithm is then used to identify the center of the star image. Basically, centroiding has three types of algorithms: Centre of mass and weight, which uses the number of pixels, and light intensity to estimate the center point of a star. The accuracy is up to 1/10 of a pixel. The Gaussian distribution method produces an accuracy of 1/100 of a pixel, but it's complex to implement. Therefore, the Centre of Mass or weight is easier to implement and achieve reasonable accuracy [18]. Centroiding without defocusing would be advantageous

for achieving higher SNR and to overcome aberration issues. Using the centroid coordinates of star images, the identification algorithm is implemented.

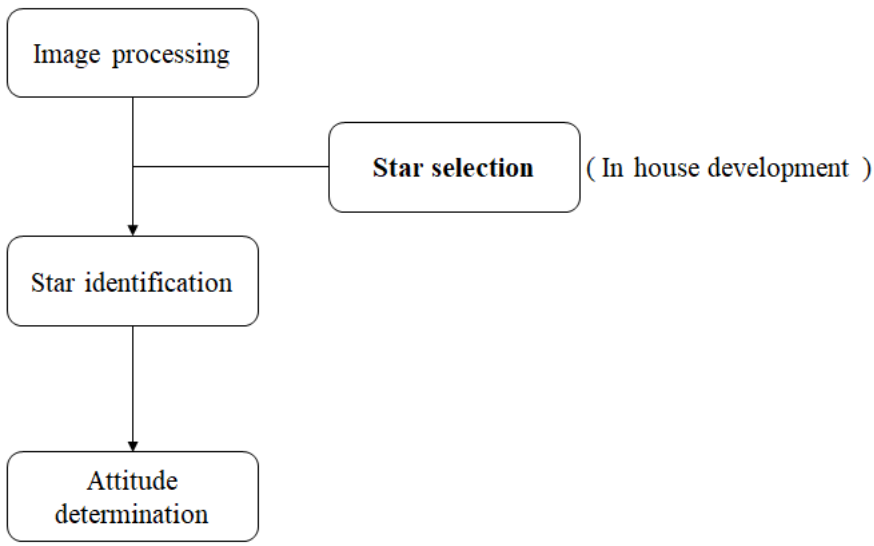

Figure 8: Algorithm flow of typical star tracker

The identification algorithm identifies stars in the image by matching them with the onboard star catalog. The identification algorithm contains two main classifications: pattern recognition and sub-graph. The pattern recognition is based on the patterns produced by connecting the stars (e.g., star constellations) using grid algorithm or ring algorithm, but the pattern recognition must have higher star density, which means more faint stars. The subgraph method uses the distance and angle information between the stars for identification. Table 5 contains a list of important identification algorithms and their characteristics [19]. The main characteristics of an efficient identification algorithm are being highly robust, less complex, having a small database, and requiring less time for execution. Comparing the four algorithms, geometric voting is more suitable for APST

because it is highly robust, has a high success rate and require an only small database, but one disadvantage is it requires more time for execution when comparing to other algorithms. The second option would be a pyramid algorithm (K-Vector search) due to its well-known success rate, robustness and shorter execution time. Grid algorithm or any other based on pattern recognition would be the option for future development because pattern recognition requires a lower percentage of trigonometrical information of stars. This could be a serious problem due to an aberration in lens, noises, and stray light and this can be overcome using pattern recognition but it requires higher star density. Further work on developing optimized pattern-based algorithm would be efficient. The Triangle algorithm is less complexity and smaller database but it does not have a high success rate.

Table 5: Comparison of various identification algorithm

Parameters	Triangle	Pyramid	Geometric voting	Grid
Robust	Low	High	High	High
Complexity	Normal	High	Normal	Normal
Database	Small	Small	Small	large
Accuracy	Sub-pixel	Sub-pixel	Sub-pixel	Pixel
Time	Less	Less	Normal	Less

Once the stars are identified, the corresponding star vector in International Celestial Reference Frame (ICRF) and sensor frame is used to estimate the attitude of the satellite. The commonly used attitude deterministic algorithm are TRIAD, quaternion and optimal algorithm are QUEST, ESOQ2. The TRIAD is simple, faster and it requires only two vectors to determine the attitude. But the drawback is more than two-star vectors cannot be used for attitude determination. The optimal quaternion and QUEST estimators consume more time but estimate the

accurate solution. The modern ESOQ2 estimators are faster and accurate [20].

1.2 Thesis Objective

The star tracker contains three main components image sensor, lens, and baffle. The reliability, accuracy and update rate of the star tracker is based on these three components. The COTS components have several advantages but they are not customized for a particular operation and if it's not selected. But there is no established approach to select the image sensor, lens and baffle form COTS. Hence methodological approach to select the image sensor, lens from COTS components is established. The miniaturized baffle is designed and manufactured in-house. But COTS component decreases the overall performance. In APST, the lens distortion increases the measurement error which causes misidentification of stars or higher processing time for identification. To overcome this, the relative star selection method is developed which selects the stars with less measurement error. The APST is positioned in SNUSAT-2 based on the estimated sun and earth avoidance angle.

1.3 Research Contributions

A Methodological approach to develop pico star tracker using fully COTS components is established. The research contribution contains two sections,

1) Theoretical model for hardware design using COTS
    - Lens and image sensor selection from COTS
    - Miniaturized baffle design and manufacturing

- Star tracker placement based on sun and earth avoidance angle

2) Measurement error tolerant star selection algorithm

- Novel and faster relative star selection method for pico star tracker
- Selection of stars with lower angular measurement error
- Relative star selection algorithm with high success rate

1.4 Research Outline

The structure of the thesis is comprised of five sections,

1) Introduction

   This chapter introduces small satellites and SNUSAT-2. The overview of current pico star tracker in market and algorithms of the star tracker is given. The research objective and contribution are showcased.

2) Hardware Selection and Development

   The theoretical model for hardware design using COTS components is described. The model contains Field of View and Limiting magnitude analysis. Followed by the image sensor and lens validation using the signal to noise ratio estimation. Miniaturized baffle design and manufacturing are detailed. Sun and earth avoidance angle for the APST is shown. The night sky testing method is explained. (The contents in this section are published by the author under the tittle “Development of the Arcsecond Pico Star Tracker (APST)” Transaction of Japanese Society for Aeronautical and Space Science, Vol. 60, No. 6, pp 335- 365, 2017)

3) Star Catalog generation

The HIPPARCOS catalog is used for APST. In this section, five steps for the generation of star catalog is described. The important things to be considered for star catalog generation based on the star tracker requirement is explained.

4) Algorithm development

The overview of algorithms used in APST is explained. The angular distance measurement error of APST is described. The relative star selection method concept is detailed and its compared with the bright star selection method. (The contents in this section are published by the author under the tittle “Star Selection algorithm for Arcsecond Pico Star Tracker” AIAA SciTech Forum, 10.2514/6.2018-2199, Kissimmee, Florida, January 8-12, 2018)

5) Simulation results

The star tracker simulator is explained and simulation results containing the accuracy and speed of the relative star selection and bright star selection algorithm is explained. (The contents in this section are published by the author under the tittle “Star Selection algorithm for Arcsecond Pico Star Tracker” AIAA SciTech Forum, 10.2514/6.2018-2199, Kissimmee, Florida, January 8-12, 2018)

6) Conclusion

This chapter includes the concluding remark and suggested future work.

Chapter 2: Hardware Selection and Development

The development of APST contains two section hardware development, validation and software development, and validation. In this chapter hardware, development and validation are described and it contains a number of steps which is shown in Figure 9. First, star tracker requirements are defined. Second, required FOV and limiting magnitude star is estimated. Based on the two previous steps, image sensor and lens are selected from COTS. Using a signal to noise ratio model selected components are validated theoretically. In flowing the night sky testing validates the components experimentally. Based on the image sensor, lens selection, and system requirement the baffle is designed and manufactured. The algorithm development, simulation test results are discussed in chapter 4 and 5.

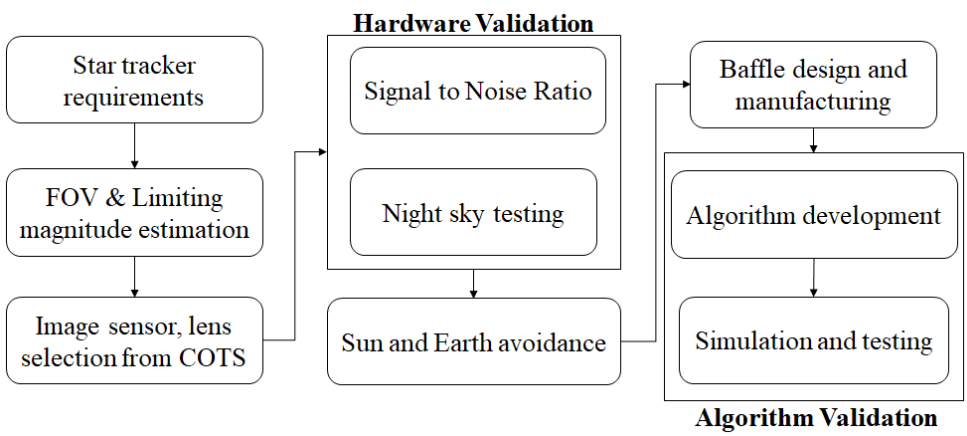

Figure 9: Development process of APST

The APST development starts with the requirement. Based on the literature review limitations of nanosatellites and SNUSAT-2 mission, the important requirements of APST is listed in Table 6.

Table 6: Requirements of APST

Parameters	Requirements
Weight including baffle (g)	< 150
Size including baffle (mm)	48x48x90
Nominal Power (mW)	< 500
Accuracy 3σ (arcsecond)	< 50 (PY), < 200 (R)
Update rate LIS (Hz)	1

2.1 Field of View and Limiting Magnitude Estimation

The High Precision Parallax Collecting Satellite (HIPPARCOS-2) catalog is used for APST. The catalog contains 118,218 stars in total, and the accuracy of star magnitude is up to a factor of 4. In APST, the stars of magnitude brighter than 6 are used [21]. Hence, for initial analysis, only 4,558 stars are used. Figure 10 shows the number of stars brighter than the magnitude of 6. The number of stars increases exponentially as the apparent magnitude of the star increases. The stars are non-uniformly distributed over the sky because the star density is higher in the galactic plane and lower at the galactic poles. Figure 11 shows the distribution of stars of magnitude brighter than 6 in June 2016, clearly implying that the stars nearer the poles have a relatively lower magnitude compared to those nearer the equator. The average density of stars brighter than the magnitude of 6 over the entire sky, galactic plane, and galactic poles is 0.15, 0.32, and 0.13 per square degree, respectively [22].

The FOV of a star tracker is a crucial factor that determines the requirements for the image sensor, optics, and baffle. A star tracker with a WFOV ranging from 15° to 40° is the optimum range for the nanosatellites. The WFOV has advantages like lower memory, less processing time, and brighter stars. In general, star tracker

needs two stars to determine the attitude of the satellite. We use pyramid algorithm that applies the angular distance between stars to identify stars. The pyramid algorithm requires minimum four stars in the FOV. When there are only two or three stars in a FOV, there is a high probability that stars misidentified or unidentified. If the angular measurement error of the star tracker is less than 30 arcsecond and there are a minimum of four stars in the FOV, then all the four stars can be successfully identified. But there is always a possibility of false stars being in a FOV and to compensate for this, we assume that minimum five stars are required for the FOV estimation.

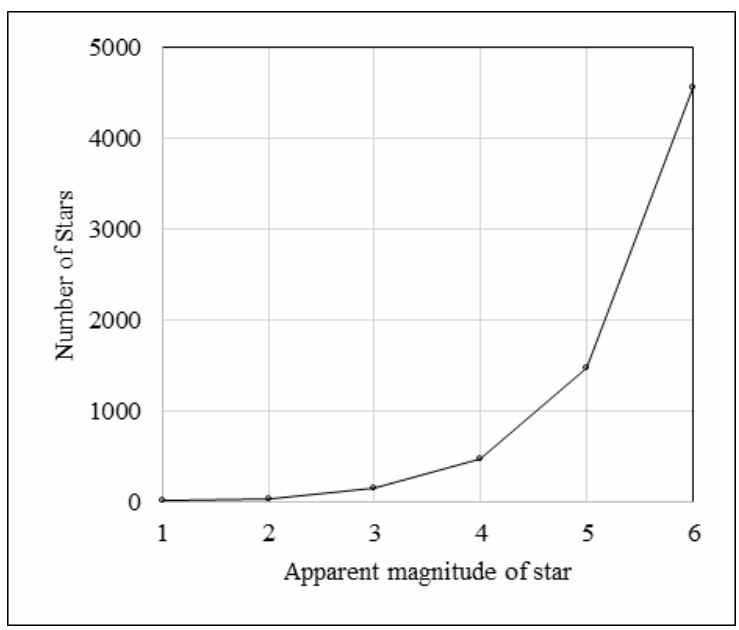

Figure 10: Number of stars corresponding to the apparent magnitude

The larger the number of stars in the FOV, the higher the accuracy and success rate; however, this consumes more operation time. Therefore, we should make a trade-off based on the accuracy and update rate requirement. The APST requires having

a minimum of 99% sky coverage containing a minimum of five stars in a FOV to determine the attitude of the satellite.

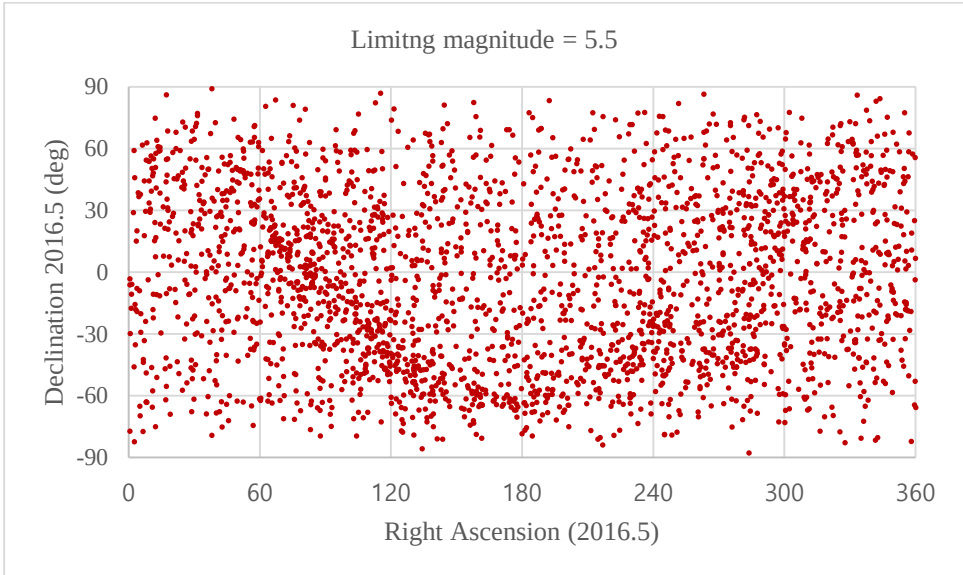

Figure 11: Distribution of stars in the sky

The simulations are performed in MATLAB to determine the required FOV containing minimum five stars in any part of the sky. The stars of brighter than a magnitude of 6 are used for this simulation. The Circular FOVs (CFOVs) of 17°, 20°, 22°, 25°, 30°, 35° and 40° are used for analysis. In order to estimate the FOV, a unit celestial sphere is created using real star coordinates and the star tracker is assumed to be in the center of the unit sphere. The simulation also generates a random location on a unit sphere which represents the pointing direction of the star tracker.

Then the number of stars present in the FOV is estimated for 10,000 iterations, which are randomly selected sky locations. This program performs many iterations to collect statistics about how many stars lie within the FOV of the

star tracker. The error in the approach is $1/\sqrt{N}$, where N is the number of iterations. Hence, the result contains an error of 1 in every 100 random locations. This means the higher the number of iterations, the higher the accuracy of identifying the number of stars in the location. The simulation method and codes for FOV estimation were made by "Scott Mulligan" and readers can go through for the details of this simulation and code [23].

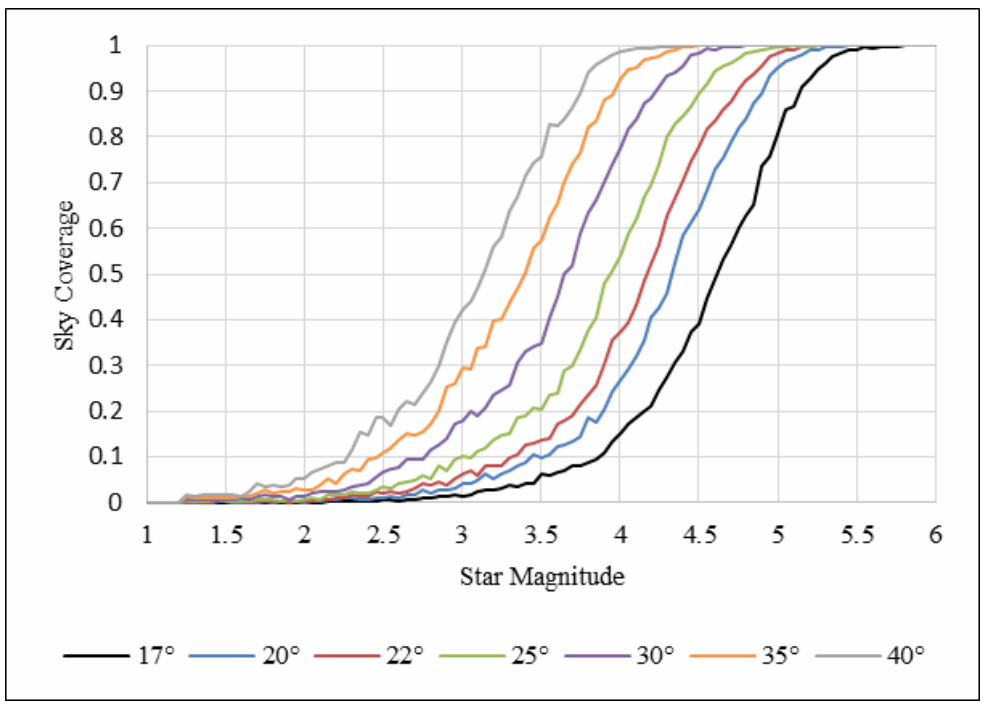

Figure 12: Sky coverage for various FOV and required star magnitude

Figure 12 shows the required star magnitudes for different FOV and their corresponding sky coverage. Based on the required sky coverage, the limiting magnitude can be determined. The minimum sky coverage required for APST is 99% and the maximum is 100%. Table 7 shows the required limiting magnitude for sky coverage of 99% and 100% for various CFOV. To acquire 100% sky coverage, the

CFOV of 17° requires a limiting magnitude of 5.85, whereas the limiting magnitude is 4.4 at 40˚.

Table 7: Required limiting magnitude for sky coverage

Circular Field of View (degree)	Limiting magnitude for sky coverage of 99%
17	5.5
20	5.25
22	5.1
25	4.9
30	4.55
35	4.35
40	4.05

The wider FOV requires the minimum of a number of stars and processing time can be significantly reduced. However, wide FOVs of 40° and 30° is more vulnerable to stray light from the sun, earth, and moon. Therefore, the CFOV of 17° and 22° are selected for initial analysis because they are less vulnerable to stray light and relatively accurate. But many star tracker developer used WFOV 30° to 40°. The selection of imaging sensor and optics for a star tracker should be based on the FOV and limiting magnitude requirement. The image sensor and optics of the APST are selected based on the CFOV 17° and limiting magnitude of 5.5.

2.2 Selection of Image Sensor and Optics

The image sensor contains various parameters that determine its function and performance. The active pixel sensor (APS)-based CMOS sensor is preferred instead of a CCD due to the advantages of windowing, lower power consumption, and lower price. The fabrication of the radiation-tolerant CMOS APS image sensor using standard CMOS processes provides a considerable cost advantage over other

image sensors fabricated using specialized radiation tolerant processes. Moreover, other radiation tolerant electronics can be integrated with CMOS APS image sensors using the same design and standard CMOS fabrication process. This enables miniaturization of the radiation tolerant imaging system. This advantage makes CMOS APS a viable alternative to CCD for space applications.

The image sensor can either be monochrome or color. Monochrome is preferred because its quantum efficiency is higher than a color sensor, and a star tracker does not need color information on the star. The usage of a color sensor for star tracker will be one of the research topics in the future. Table 8 shows three CMOS-based monochrome image sensors and the important parameters to be considered [24]. The sensor size of 1/3″ to 1/2″ is suitable for APST. The pixel size is the important factor in determining the accuracy of the sensor; the smaller the pixel size, higher the accuracy. The star tracker usually has an image sensor with a resolution less than 1 MP.

Table 8: Important parameters of image sensor

Parameters	MT9P031	AR0134	Python 1300
Sensor size (inch)	1/2.5	1/3	1/2
Pixel size (μm)	2.2	3.75	4.8
Resolution (MP)	5	1.2	1.3
QE at 525 nm (%)	63	77	59
Sensitivity (V/lux.s)	1.4	7.7	6.1
Read noise (e-)	7.64	6.58	9.28
Full well capacity (e-)	6693	5542	6057
Dynamic range (dB)	58.3	64	55.84
Shutter	Rolling	Global	Global

The problem with a high-resolution sensor is it takes a longer processing time, which will reduce the update rate of the star tracker. The higher the quantum

efficiency, the lower the read noise and dark noise is better because it enables the imaging of faint stars. The full well capacity determines the brightest star it can image. The dynamic range is the ability to image the brightest and faintest star in the sky. The star tracker needs an images sensor with higher dynamic range. The CMOS sensor usually has a rolling shutter but the latest CMOS sensors have a global shutter. The global shutter can image without smearing even if a satellite maneuvers at a relatively high rate, whereas the rolling shutter will smear the image. Therefore, the global shutter is preferred over the rolling shutter. The final important thing is the availability of the image sensor in the market. The first preference is the AR1034 because among the three it has the highest sensitivity, quantum efficiency, and dynamic range. However, only the MT9P031 is the only one available, and it satisfies most of the requirements.

Table 9: Important parameters of imaging lens

Parameters	BHR 16012	BL 16014	B3M 16018	BSM 12016
Optical format (inch)	1/2″	1/3″	1/2″	1/2″
Focal length (mm)	16	16	16	12
Focal ratio	1.2	1.4	1.8	1.6
FOV (deg)	15	15	15	19.6
Resolution (MP)	1	1	3	<1
Resolution (μm/line)	8.98	5.96	5.18	9.21
Weight (g)	12	16	--	5

The MT9P031 has already been used in the ST-16-star tracker developed by University of Ryerson, Canada and have been successfully operated in many missions since 2013. Hence the MT9P031 monochrome image sensor was selected for APST. Based on the MT9P031 image sensor, an imaging lens was selected. The important parameters of imaging lens are shown in Table 9. The S-

mount lenses are chosen because they weigh less. The lens with a low focal ratio (1.2 to 1.8) maximizes the light collection, but aberration increases rapidly as the focal ratio decreases. Spherical and chromatic aberration in the lens may cause failure during the execution of star identification. Equations (1 to 3) shows the relationship between the aberration and focal ratio. Equations 4 and 5 are used to estimate Field of View and sensor pixel accuracy respectively. The focal length of 16mm and 12mm provides the FOVs of 15˚ and 20˚, respectively. Additionally, the focal length determines the accuracy of the star tracker; higher the focal length the better the accuracy of the star tracker. Using 16 and 12 mm focal lengths, the theoretical pixel accuracies of 28.4 arc-sec and 37.8 arc-sec are obtained, respectively.

$$Spherical\ aberration\ \propto \frac{1}{(focal\ ratio)^3} \quad (1)$$

$$Coma\ \propto \frac{1}{(focal ratio)^2} \quad (2)$$

$$Astigmatism\ \propto \frac{1}{(focal\ ratio)} \quad (3)$$

$$Field\ of\ View\ \propto \arctan\left(\frac{sensor\ size}{focal\ length}\right) \quad (4)$$

$$Pixel\ accuracy\ \propto \arctan\left(\frac{pixel\ size}{focal\ lenght}\right) \quad (5)$$

The resolution of the image sensor lens should be in high- resolution MP, otherwise, the image that is output will be blurred. This means the magnitude of the signal is reduced in the image sensor. A lens with a resolution of 1 to 3MP is chosen. Theoretically, the 1MP and 3MP lenses can resolve at 8.9 μm/line and 5.9 μm/line, respectively. These characteristics will be studied in detail during laboratory and night sky testing. Based on the testing results, the B3M16018 lens in

Table 9 is chosen. Distortion is one of the important factors in determining the quality of the lens. There are three types of distortion: barrel, pincushion and mustache. The distortion correction algorithm is required in the post-processing to overcome the error due to distortion in the lens. This enables distortion to be compensated. The B3M16018 lens has a pincushion distortion of 0.65%. Based on the suggestions from experts, we opted for a lens with less than 1% distortion. An antireflection coating for the lens increases transmission of the light. An imaging lens with filters and customized lenses would enhance the quality and performance of the star tracker, but these are expensive and will be considered for development in the future.

2.3 Signal to Noise Ratio Estimation

The previous sections have established the required FOV, available image sensor, and optics. The signal to noise ratio (SNR) of the APST is one of the factors for verifying if the image sensor and imaging lens meet the FOV requirements. If the APST has a minimum SNR of 8, it can easily detect stars. The signal of the APST is estimated using Equation 4, where R is the radius of the lens, t is exposure time (0.1s), F0 - 21,161 ph/s/mm^2 is the theoretical flux of the zero magnitude star detected by the MT9P031 image sensor (based on the quantum efficiency of the image sensor) [25], m is the limiting magnitude of the star tracker (m= 5.35, 5.85)]. The noise estimation of the MT9P031 image sensor is shown in Table 10. A detailed method for estimating the noise of the APS-based image sensor is given [26] [27] [28] and estimated using Equation 7.

$$Signal = \frac{\pi R^2 tF}{2.5^m} \quad (6)$$

$$Noise = \sqrt{(S + DC + DCNU + R^2 + Q^2)} \quad (7)$$

The number of pixels included in the star image is based on the Point spread function (PSF), by increasing the PSF, the number of pixels that contribute to the noise increases, whereas the magnitude of the signal decreases. The symmetric PSF is considered for analysis, and due to the slew rate of the satellite, the noise in the pixel's increases, which is a function of focal length. The total number of pixels in the PSF is estimated using Equation 8 [28]. Where P is the PSF radius in pixels, F is the focal length (mm), ω is the slew rate (0.1° /s), t is the exposure time (0.1s) and γ is pixel size (2.2 μm). The SNR for various PSF radii and for the corresponding focal ratio of the lens is estimated in Equation 9.

Table 10: Noise estimation of MT9P031 image sensor

Parameters	Noise (e)
Dark current (D)	0.39
Dark current non-uniformity	0.04
Read noise (R)	3.5
Quantization noise (Q)	0.46
Photon shot noise (S)	√Se

$$Number\ of\ Pixels = \sqrt{\pi p^2 + 2p\left(\frac{F\tan(\omega t)}{\gamma}\right)} \quad (8)$$

$$SNR = \frac{Signal}{(Noise)\,(Number\ of\ Pixels)} \quad (9)$$

The graph in Figure 14 implies that the PSF radius of 0.5 has higher SNR when compare to a PSF radius of 3. Focal ratio is another important factor

increasing the SNR, the lower the focal ratio the better than signal, so it has higher SNR. The stars can be easily detected with an SNR of 8.

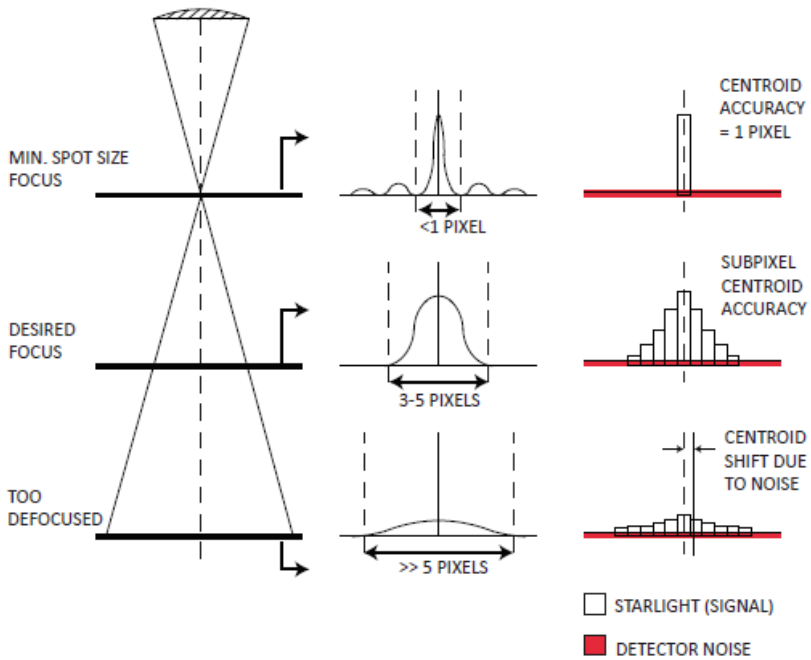

Figure 13: Variation in centroiding accuracy due to focusing and defocusing [29]

The analysis shows that the lenses with a focal ratio of 1.2 to 1.8 have an SNR higher than 8 for the PSF radius of 0.5 to 3. The focal ratio of 1.2 with a PSF radius of 0.5 has the highest SNR, 84.6, and whereas the focal ratio of 1.8 and PSF radius of 3 has the lowest SNR, 13.4. The higher the PSF radius, the better the centroiding ability; however, the higher PSF reduces the SNR, which leads to the inability to detect faint stars. Therefore, SNR and PSF should be chosen based on the requirements of the star tracker. The effect of defocusing and focusing is illustrated in Figure 13 [29].

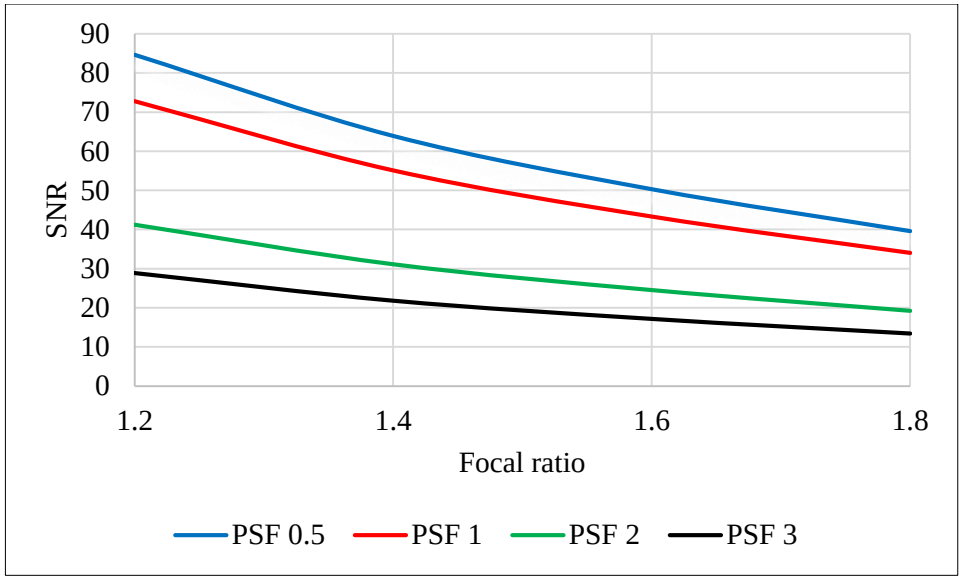

Figure 14: SNR vs focal ratio for various PSF

Table 11 shows the estimated SNR for star magnitudes of 5.85 and 5.35 using 16 mm and 12mm lens respectively. The PSF of 0.5 is when a star is focused in a single pixel and a PSF of 3 is when a star is defocused at around six pixels. In APST we prefer a focal ratio of 1.8 and PSF of 1 to 2 (PSF 0.5 is practically not achievable). For example, when imaging 5.85 magnitudes, the SNR is 72.8 (using 16 mm lens, PSF 1, and focal ratio 1.2) and the SNR decreases to 34 (using a focal ratio of 1.8) Whereas the same lens with PSF of 3 reduces SNR to 13.4.

Based on the analysis from previous sections, two possible design for the star tracker is chosen. The MT9P031 image sensor with a focal length of 16 and 12 mm have CFOVs of 17° and 22°. Based on the sensitivity analysis of APSTs with CFOV of 17° and 22° can detect stars with magnitudes of 5.5 and 5.1, respectively, which assures sky coverage of 99%. Table 12 lists the two possible designs for the APST.

Table 11: Estimation of SNR for various focal ratio and PSF

Focal length (mm)	Focal ratio	SNR PSF0.5	SNR PSF 1	SNR PSF 2	SNR PSF 3
16	1.2	84.6	72.8	41.2	28.8
	1.4	63.9	55.1	31.1	21.8
	1.6	50.3	43.3	24.5	17.1
	1.8	39.6	34	19.2	13.4
12	1.6	50.2	38.8	21.5	14.9

Table 12: Two possible design for APST

Parameters	Design α	Design β
Min. no. of stars in a FOV	5	5
CFOV (deg)	17	22
Limiting magnitude	5.5	5.1
Total no. of stars	3897	2237
Focal length (mm)	16	12
Focal ratio	1.8	1.6
Accuracy (arcsecond)	28	38
Execution time	Longer	Shorter

The both design α and β have their own advantages and disadvantages. Design α has better accuracy, but the total number of stars is higher. Therefore, the algorithm execution time would be relatively longer. The total number of stars in design β is less. Therefore, the execution time is relatively less, but accuracy is lower as well. Both of this design is suitable for an APST, based on night sky results, the design α is chosen.

2.4 Night Sky Testing

Based on the theoretical estimation of the sensitivity of the image sensor and imaging lens, an MT9P031 demonstration kit and Lensation lenses are used for the testing, which is shown in Figure 15 and Figure 16. The MT9P031

demonstration kit contains a wide angle lens, which has been replaced with a Lensation lens, using an adapter as shown in Figure 15.

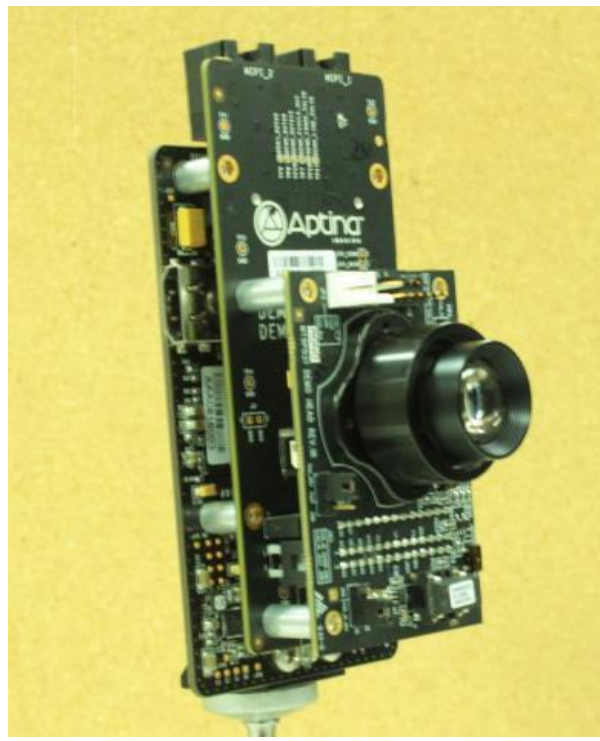

Figure 15: APST demonstration kit

The parameters of the MT9P031 image sensor and Lensation lenses are shown in Table 8 and Table 9 respectively. In this night sky testing, the hardware is tested to see if it can detect stars of 5.5 magnitudes with an exposure time of less than 200 ms to ensure APST has 99% sky coverage. But the algorithms for image processing, identification, and attitude determination have not been implemented. First-night sky testing is performed at the Seoul National University and around Seoul but, the test results were poor due to the city lights. Therefore, we traveled 160 km southeast of Seoul to Yongjin-ri which is located in the state of Danyang-gun. This location is a remote area and there are no artificial lights in the

surrounding area. Based on the Accu Weather forecast there were no clouds (i.e., 100% clear sky) on August 12, 2016, at 1:00 a.m. The humidity was above 90%, but dry weather is a better condition for star imaging.

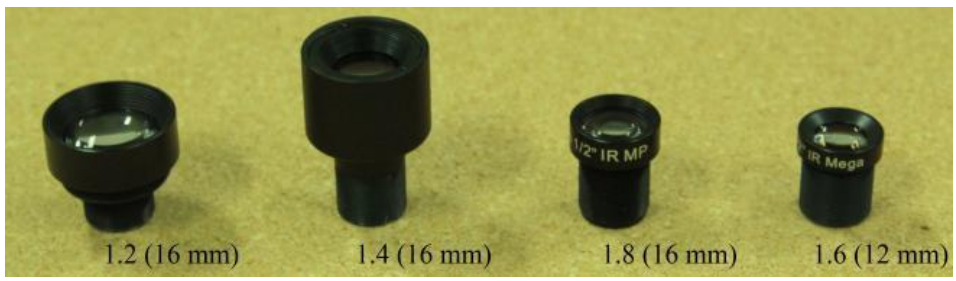

Figure 16: Imaging lenses of APST

The stars near the zenith have a brightness loss of 0.2 visual magnitude and stars near the earth surface have a brightness loss of 1 visual magnitude due to the high atmospheric air mass [30]. Therefore, stars within the zenith angle of 45° are imaged during the night sky testing, which has a brightness loss of 0.3 visual magnitude19. If APST can detect 5.3 visual magnitude stars in night sky testing, it can detect stars of 5.5 visual magnitude in orbit.

All four of the lenses in Figure 16 are tested using the MT9P031 demonstration kit for the exposure time of 200ms and using the maximum gain value. Two of the lenses are selected based on night sky testing. Frist BHR16012 is the brightest lens having a focal number of 1.2 and resolution of 1 MP. Second B3M16018 of focal number 1.8 and resolution of 3 MP. The numbers in Figure 17 to 20 shows the visual magnitude of the stars. The 1.2 focal number lens imaged stars with a visual magnitude up to 6.1. The APST only requires a visual magnitude of 5.5; therefore, it can operate with an exposure time less than 200

ms. The disadvantages focal number of 1.2 lenses are low resolution and high aberration. In Figure 18, the star Vega (zero magnitude) is elongated due to aberration (unsymmetrical). Figure 17 has a focal number of 1.8 and resolution of 3MP and the lens is not as bright as 1.2 focal lenses but it can still image a star with a magnitude of 6 using an exposure time of 200 ms. However, the 1.8 focal lens produced symmetric images without aberration and distortion as shown in Figure 17 which is better when compared with Figure 18.

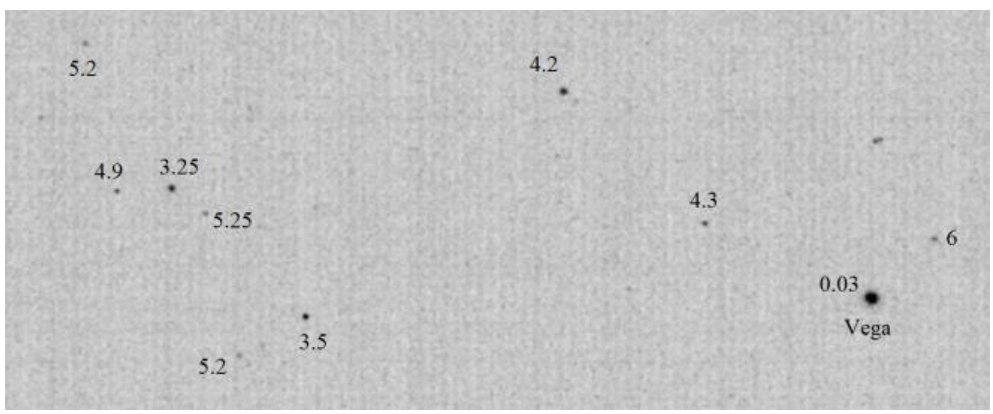

Figure 17: Lyra constellation imaged using 1.8 focal number lens

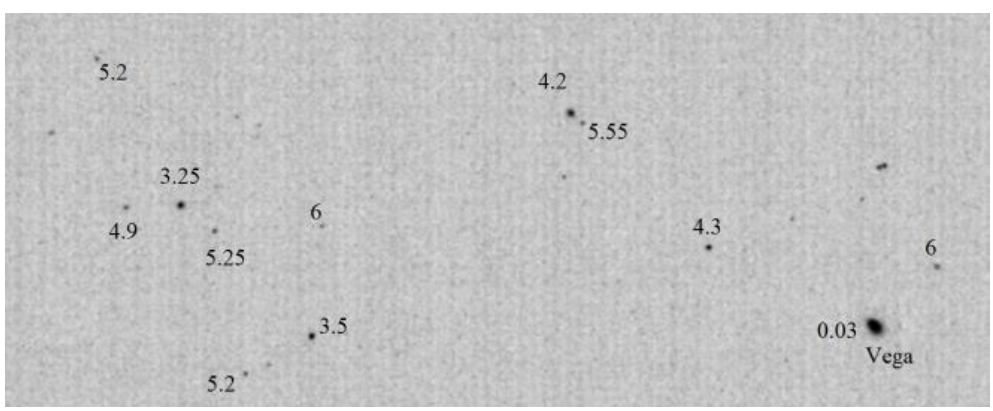

Figure 18: Lyra constellation imaged using 1.2 focal number lens

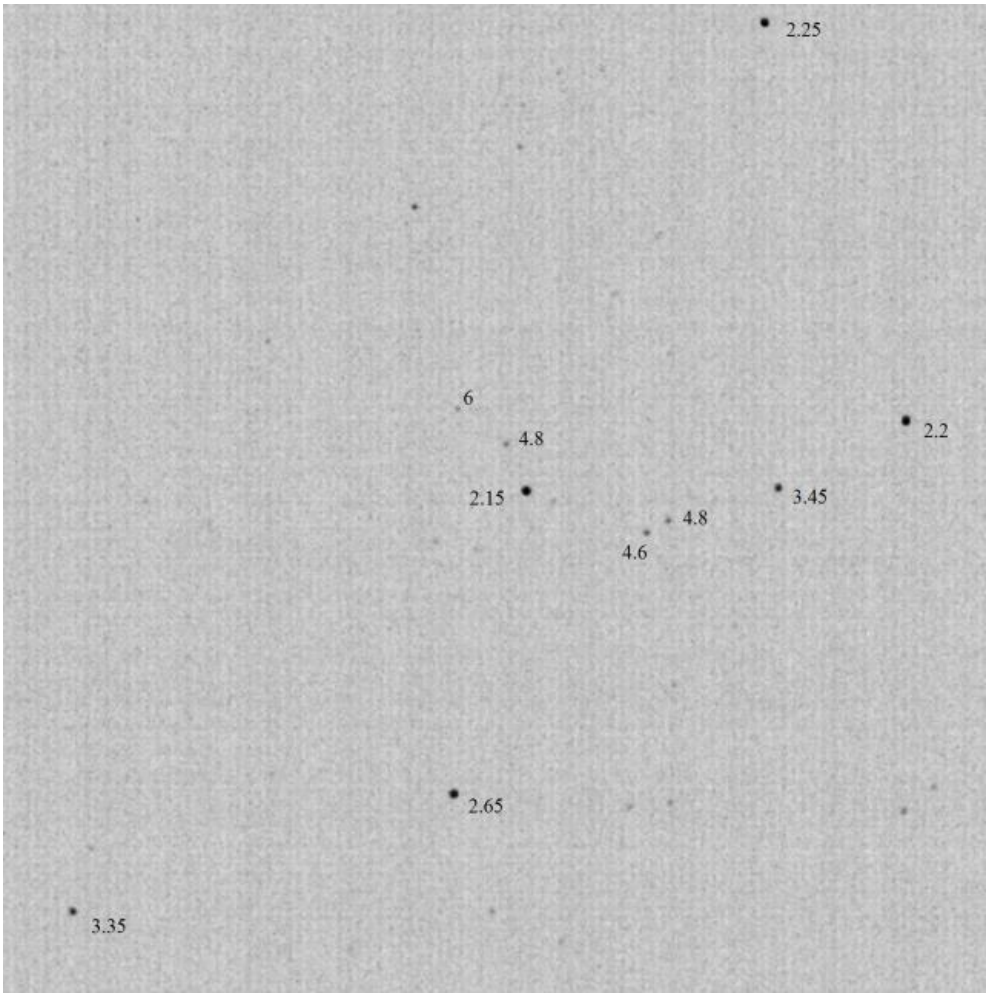

Figure 19: Cassiopeia constellation image using 1.8 focal number lens

When comparing Figure 19 and Figure 20 the stars with a magnitude of 2.15, 2.65, 3.35 are elongated in Figure 20 and almost symmetrical in Figure 19. The stars imaged using 1.8 focal number are more symmetrical than those imaged using 1.2 focal number and this will help to accurately find the centroid of the star and it leads high success rate of identifying the stars. Therefore, a lens with 1.8 focal number is chosen for APST.

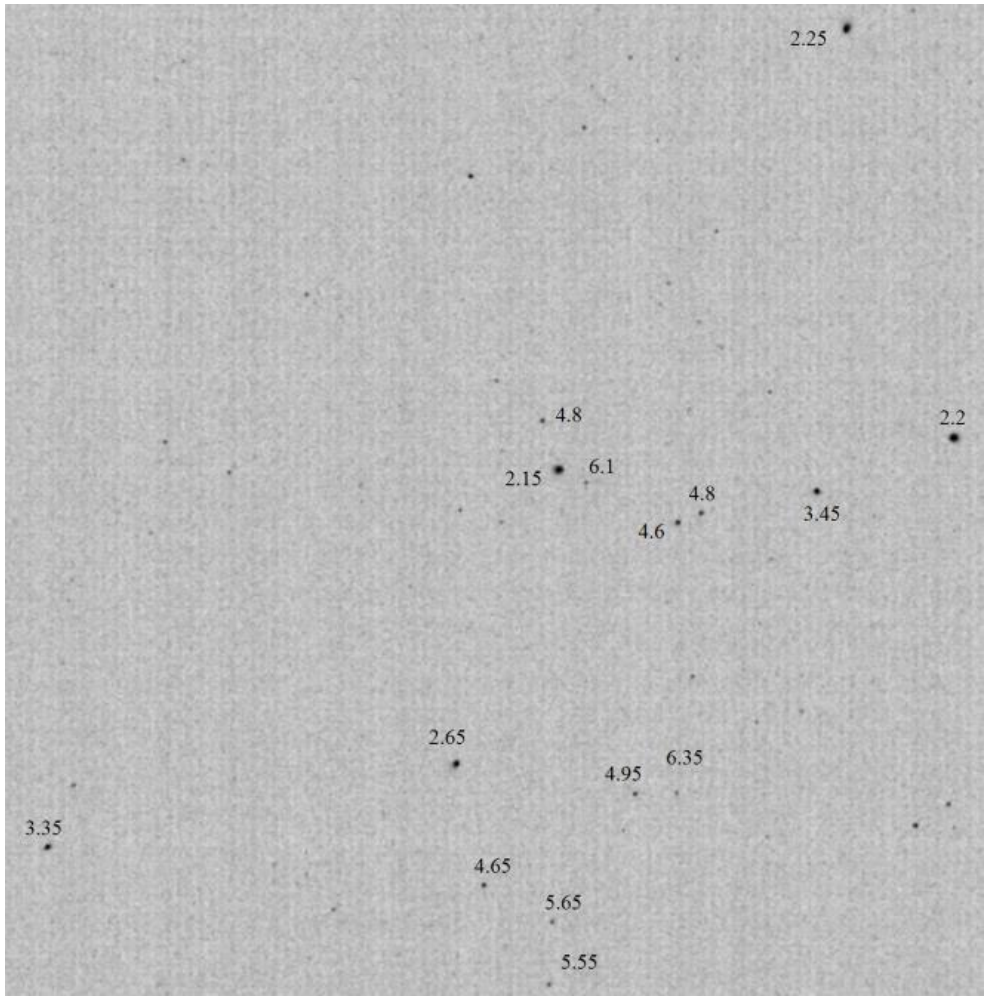

Figure 20: Cassiopeia constellation imaged using 1.2 focal number lens

In Figure 21 the star imaged (after image processing) with 1.2 and 1.8 focal number lens is shown. The star imaged with 1.2 focal number looks elongated (unsymmetrical) but the same star imaged with 1.8 focal number looks more symmetrical when compared to 1.2 focal number lens. Table 13 contains the minor axis and major axis of the stars imaged with 1.2 and 1.8 focal lens. The minor and major axis length is given in pixels. The values in Table 13 clearly show the aberration effects on 1.2 and 1.8 focal number lens.

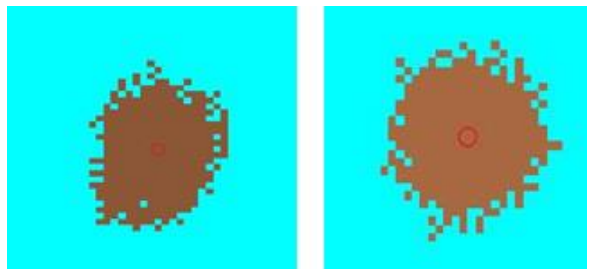

Figure 21: Aberration effect on 1.2 and 1.8 focal number lens

Table 13: Aberration effect on stars based on focal number

No.	Focal number – 1.2		Focal number – 1.8	
	Minor axis (pixels)	Major axis (pixels)	Minor axis (pixels)	Major axis (pixels)
1	17.2	10.6	20.8	20.3
2	15.2	12.3	15.7	15.3
3	13.9	11.6	14.2	14.6
4	15.5	10.8	14.5	13.7
5	11.4	7	9.5	8.4
6	15.2	7.7	9.6	9.4

These tests confirmed that existing hardware can detect stars of 5.5 magnitude with an exposure time of less than 200 ms. This shows that the APST will have 100% sky coverage during static imaging, but the sky coverage will be decreased under dynamic condition (high slew rate). This night sky testing is done without the operation of algorithms and baffle. A night sky test with various slew rate will be performed.

2.5 Sun and Earth Avoidance

The stray light from the Sun, Earth and Earth's moon is one of the major factors in lowering the performance of the star tracker. It could even lead to failure in certain conditions. As APST will be operating in a Low-Earth Orbit (LEO), the dominant stray light source is the Earth. The SNUSAT-2 orbital altitude will be

around 400 to 700 km, which is far less than the radius of the Earth. Therefore, the Earth is viewed as an extended stray light source. However, the bright stray light sources like the Sun and Earth's moon are viewed as point sources due to their long distance from the LEO satellites. To successfully image the faint stars of 5.85 magnitude, the background stray light must be lower than the magnitude of 5.85. The optical axis of the star tracker should be oriented normal to the Sun to reduce the stray light intensity. Since the Earth is an extended surface, the effective stray light region of the Earth is calculated.

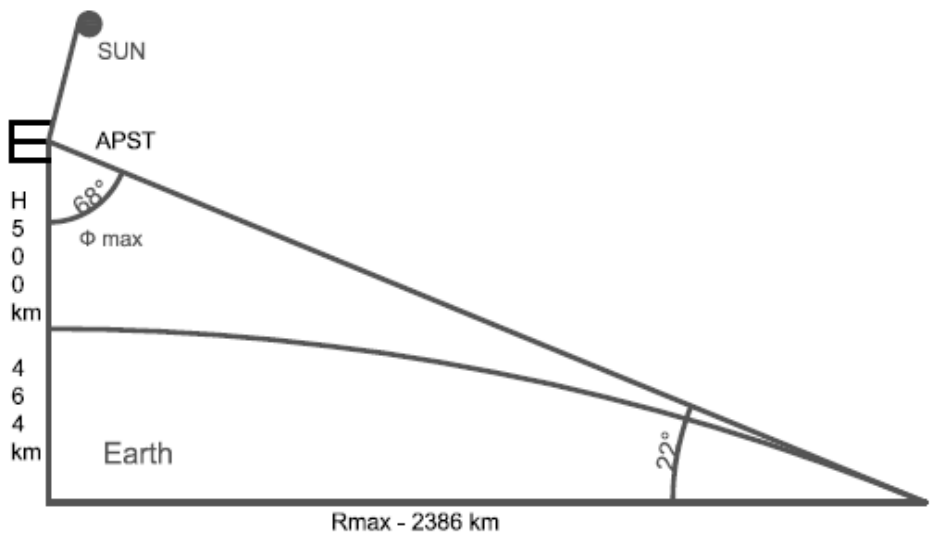

Figure 22: Extended stray light source from Earth

Figure 22 shows the orientation of APST, which is almost normal to the Sun and Earth. It also shows the effective stray light region of the Earth. The maximum stray light angle is due to the Earth incident on the APST is known as Φmax [31]. The average radius of the Earth, Rearth, is 6370 km, H is the altitude of the orbit (500 km) and the Φmax is 68°. The maximum radius of the effective stray light

region is R_{max}. The maximum irradiance from the sun is 918.1 W/m^2 and the Earth reflects 35% of the Sun's light, which is known as albedo.

$$\emptyset_{max} = \sin^{-1}(R_{earth} + H) \quad (10)$$

$$R_{max} = R_{earth} \cos \emptyset_{max} \quad (11)$$

2.6 APST and Baffle Design

The maximum irradiance from the Earth is 321.3 W/m^2. The irradiance of the Earth's moon when it's full is 1.4 mW/m^2 [32]. The function of the baffle is to prevent stray light from bright objects (i.e., Sun, Earth and Earth's moon) outside the FOV from directly reaching the lens surface and to reduce the intensity of stray light so that the star tracker can identify the stars effectively. A single-stage, diffused cylindrical baffle with straight vanes has been designed for the APST. A two-stage baffle is efficient for reducing stray, light but due to volume constraints, a single stage baffle is used for APST. Baffles can be designed in either cylindrical or conical shape.

A baffle with specular reflection is highly dependent on low reflective paint and precise machining, and because of this reason diffused baffle is easier to fabricate. The baffle with a straight vane has a lower Point Source Transmittance (PST) of 10^{-6}, whereas baffle with grooved vanes and vane less have PST of 10^{-5} and 10^{-3}, respectively [33]. It has a half FOV of 7.5° and exclusion angle of 27°. The detailed method for baffle design is explained in the reference [34].

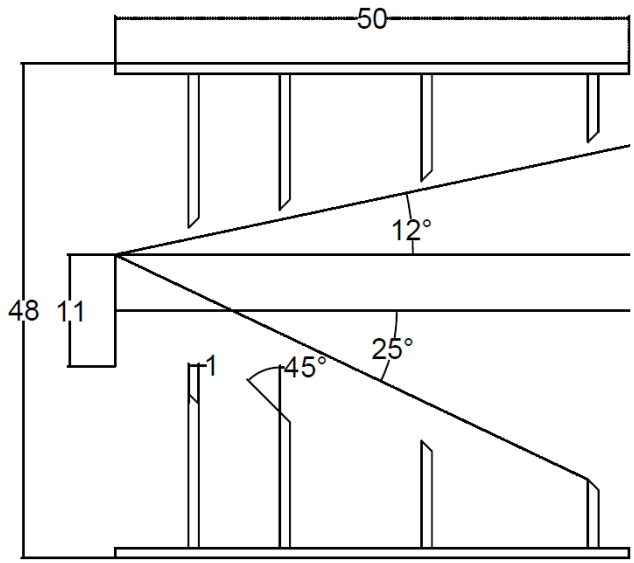

Figure 23: 2D miniaturized baffle design for APST

The baffle design should consider the following guidelines, which are described in the reference [35]. The star tracker FOV should not interfere with the baffle wall or edges. At least two reflections from a blackened surface are required between the stray light source and the optical elements. The stray light within the baffle is required to have a maximum number of reflections before it enters the sensor. A minimum number of edges should be exposed to the sun. The vanes of the baffle should have sharp edges.

The baffle is to be designed based on the required star tracker FOV and exclusion angle. The exclusion angle is the minimum angle at which light from a bright object outside the FOV can reach the lens surface. The lower the exclusion angle the better the attenuation to stray light. A baffle with higher length to diameter (L/D) ratio has a lower exclusion angle, but a baffle with a higher (L/D) ratio becomes heavier and larger. The baffle being used is designed based on the MT9P031 image sensor

and B3M16018. The baffle design for the APST, shown in Figure 23, has a diagonal FOV of 24°, baffle length of 50 mm diameters of the lens and baffle are 10.9 mm and 46 mm, respectively.

The next step is the vane placement in the baffle. The APST baffle design in Figure 23 contains four straight beveled vanes. In order to reduce the reflecting surface of the vane edge, the edges of the vanes are sharpened, which is known as beveled vanes. The vanes are designed with a constant beveled angle of 45° because it's easier to manufacture. A vane edge thickness of 0.1 mm is achievable. The Number of vanes and depth of the vanes in the baffle determine the spacing between the vanes. The number of vanes should be optimized because it enhances the risk of reflecting the stray light directly to the optics. The thickness of the baffle and vanes is 1 mm. The 3d design of the cylindrical baffle with vanes is shown in Figure 24.

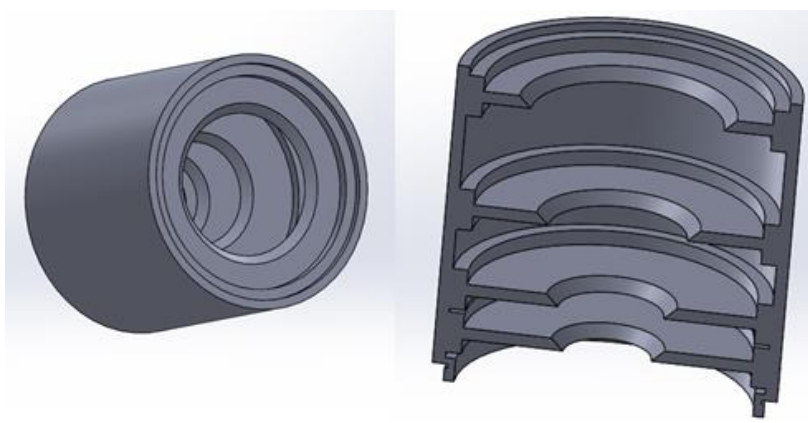

Figure 24: Design of cylindrical baffle with beveled vanes

Figure 25: Miniaturized cylindrical baffle with beveled vanes

Another important factor is the reflectivity of the black paint; the inner surface of the baffle is painted to reduce scattering of the stray light. The paints 3M black velvet, Aero glaze Z306, Martin black, Parson black, 3M 401-C10 have very low reflectivity. But due to lack of availability, black anodized is easier to obtain. Figure 25 shows the flight model of the cylindrical baffle manufactured for APST. The baffle for APST is designed to reduce the stray light effect but there are many limitations for manufacturing.

The current baffle design is complex to manufacture and the manufacturing price is high. Hence we bought a cylindrical pipe of 54 (L) 50 (ID)mm with inner thread form Thor labs. The four beveled vanes of 2mm thickness with outer threads are manufactured in the local lathe shop. Then beveled vanes are screwed into the cylinder and the inner edge of vanes are glued with epoxy for rigidity. The epoxy which we used is space proven, hence it doesn't produce out gassing in vacuum conditions.

Figure 26: Arcsecond Pico Star Tracker

Figure 26 shows the inner structure of the APST design. The APST contain two electronic circuit board. The top PCB contains the MT9P031 image sensor and ADT7410 temperature sensor, the B3M16018 lens is placed on s-mount and glued with epoxy for rigidity. The bottom board contains the STM32F429ZIT6 processor, IS42S16800J SDRAM, and SD card. The dimension of PCB is 48×48mm and thickness are 1.6 mm, without the baffle the dimension is 37.5 x 48 mm.

Figure 27 shows the APST flight model without and with baffle. The four side panels are manufactured with 2mm thickness, top and bottom panel has a thickness of 4 and 6 mm thickness. The baffle is connected with the APST via the internal threads in the top panel and the PCB stack is connected to the outer panels using M3 screws. All the external panels and baffle is manufactured using Al6061 and

black anodized. The anodizing reduces the effect of stray light scattering. The dimension of APST without baffle is 61×61×44 mm and with baffle 61×61×102 mm.

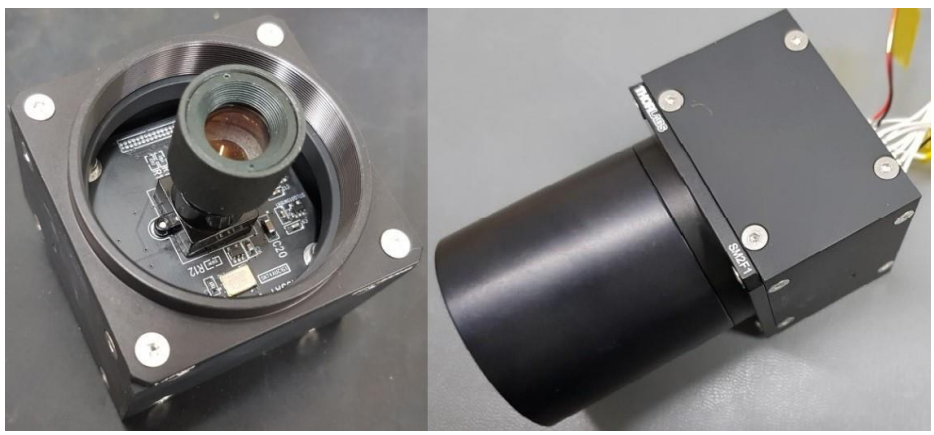

Figure 27: APST without and with baffle

Figure 28 shows the 3d model of the inner structure of the SNUSAT-2 design. The APST is tilted 10° in order to avoid extended stray light from earth. The APST is connected with SNUSAT-2 using two frames, one on the side panel and other using baffle holder. The outer side of the baffle is covered with aluminum sheet. Figure 29 shows the inner structure of SNUSAT-2 flight model and position of APST flight model in the satellite.

The system level vibration, thermal vacuum test and thermal baking are performed for SNUSAT-2. The APST performed well in vibration and thermal vacuum test without any breakage or failure. The APST is faced towards outer space in SNUAT-2 to avoid the sun and earth, hence there will be not much variation in temperature.

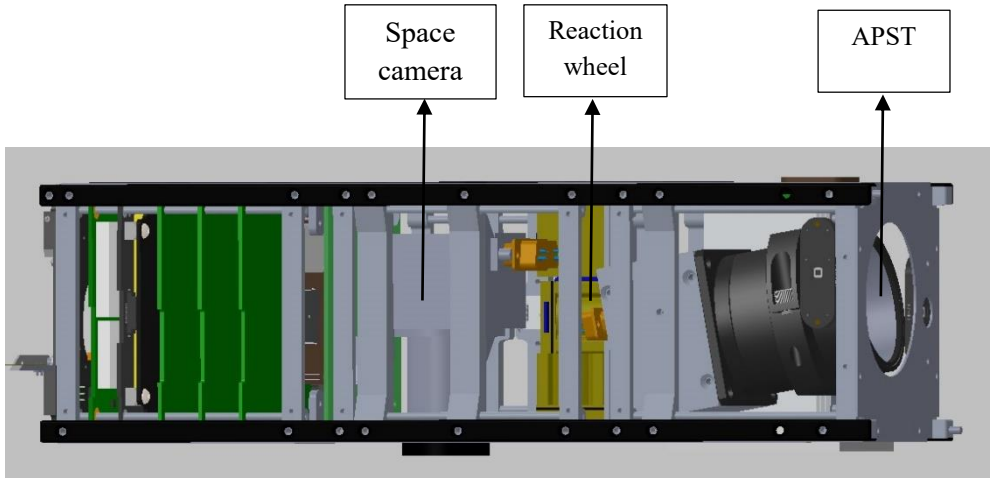

Figure 28: APST placement in SNUSAT-2 CAD model

Figure 29: APST placement in SNUSAT-2 flight model

The APST is planned to be operated in a Sun-synchronous polar orbit. The altitude of the orbit has not been fixed but it will be around 400 to 600 km. The electronics in a satellite are always vulnerable to charged particles from radiative sources. The energy, flux, and fluency of these charged particles vary based on the altitude and inclination of the orbit. Due to the radiation, electronics in satellite malfunctions and can even lead to functional failure. This is overcome by using

radiation-hardened devices. However, due to high cost and limited availability, it's not accessible for nanosatellites.

The COTS components are accessible for Nano-satellite, but they are not qualified for the space environment. Therefore, the components are qualified by radiation ground testing to determine the lifetime in the target orbit. The main radiation sources are trapped charged particles in Van Allen's Belt which extends from 300 to 36,000 km. The South Atlantic Anomaly is one of the main problems faced by satellites in LEO. Next, solar particle events like solar flares and coronal mass ejection occurrence are based on an 11-year solar cycle. Finally, galactic cosmic rays which are dominant over the poles due to the weak magnetic field. In SPace ENVIronmental Software (SPENVIS), all of these radiative sources are given as inputs and the radiation exposure in the target orbit is estimated. The ionizing particles lose their energy when traveling through matter and the energy is deposited in the matter. Over time, the charge accumulates, and this is known as the total ionizing dose (TID). The TID for silicon for a period of one year at the orbital altitude of 400 to 600 km and inclination of 90˚ is estimated using SPENVIS.

Table 14: Total mission dose for one year

Aluminum shielding thickness (mm)	Total Ionizing Dose (Krad)
0.0001	3040
1.4	5.9
2	3.04
3	1.39

Table 14 shows the TID accumulation for different shielding thicknesses over one year. By increasing shielding thickness, the accumulated dose is reduced; however,

due to weight constraints, a thickness of 2 to 3 mm is feasible. Based on the literature review, at high inclination orbit (705 km, 98°) with shielding of 2.54 mm, the component accumulates a TID of 4 Krad and this data is close to our estimation using SPENVIS. In general, COTS components have a radiation tolerance of 1 – 10 Krad/year and silicon have a dose limit of 5 Krad/year [36]. These numbers convey that, the APST with (Al) shielding of 3mm can survive in orbit for a period of one year.

Chapter 3: Star Catalog Generation

The star catalog is the main reference for the navigation of our ancestor. The star catalog was compiled from B.C by various civilization like Persians, Babylonians, Greeks, Indians, and Chinese. In early days, people explored new lands and migrated to new places by star navigation. In the modern science era, space agencies like NASA, ESA have developed their own star catalog by astrometry missions with higher accuracy. There is many available star catalog in online, HIPPARCOS, HD, SAO, PPM, Guide star. We use HIPPARCOS-2 catalog for APST. The HIP star catalog was gathered by ESA's astrometric satellite mission HIPPARCOS. The HIP-1 catalog was published in 1997 which contains 118, 218 stars with limiting magnitude of 12.4. The HIP-2 contains the updated version which was published in 2007.

Table 15: Important parameter for star catalog generation

HIP ID	Magnitude	RAJ2000 (deg)	DecJ2000 (deg)	pm_RA (mas/yr)	pm_Dec (mas/yr)
82	5.869	0.269092	-48.809873	-18.36	-5.82
107	5.6243	0.333832	-50.337371	7.88	11.4
122	4.9354	0.398757	-77.065725	-57.3	-177.06
124	5.6725	0.404229	61.222802	-3.14	-0.74

The HIP catalog contains many parameters, but we have to extract only the required information for our operation. First limiting magnitude, based on our mission we need stars of brightness up to 5.5. Second, Right Ascension and Declination of a star in J2000 epoch. Third, the magnitude of the star and corresponding HIP ID. Four proper motion in RA and Dec. These parameters are listed in Table 15.

The coordinates of stars are given RAJ2000 and DecJ2000 epoch. But we need the star coordinates for the required year. The stars move laterally in arcsecond for every year which is known as proper motion. This proper motion is added to RAJ2000 and DecJ2000. Equation 12 and 13 shows the position of stars for RA2017 and Dec2017. Table 16 show RA and Dec of stars in 2017 in ICRS.

$$RA2017 = RAJ2000 + \left(\frac{PM\_RA}{3600000}\right) \times 17 \quad (12)$$

$$Dec2017 = DecJ2000 + \left(\frac{PM\_Dec}{3600000}\right) \times 17 \quad (13)$$

Table 16: HIPPARCOS star catalog for 2017

Star ID	RA 2017 (deg)	Dec 2017 (deg)
1	0.398486417	-77.06656112
2	0.4561261	-3.027548617
--	--	--
2645	359.828588	6.862791544
2646	359.9792993	-65.57724138

$$\begin{pmatrix} Vx \\ Vy \\ Vz \end{pmatrix} = \begin{pmatrix} \cos\alpha \cos\delta \\ \sin\alpha \cos\delta \\ \sin\delta \end{pmatrix} \quad (14)$$

Table 17: Mission catalog

Star ID	Vx	Vy	Vz
1	0.223813554	0.001556626	-0.974630735
2	0.998572612	0.007949707	-0.052816105
--	--	--	--
2645	0.992830708	-0.002970264	0.119492107
2646	0.413466106	-0.000149383	-0.910519499

The star position is represented in a unit vector in the earth-centered inertial reference frame, star tracker uses these star position as a reference to determine its boresight direction. But the star coordinates are represented in RA (α) Dec (δ),

hence it's converted in unite vectors using Equation 14 [37]. The unit vectors of stars are shown in Table 17. Based on the limiting magnitude of 5.5, there are 2654 stars in total and eight stars are removed (e.g., binary stars, variable stars). The possible star pairs out of 2646 stars are 3499335-star pairs. The angular distance between two stars i and j in the catalog is calculated using Equation 15. Since Diagonal Field of View (DFOV) of APST is 24˚, the star pairs with an angular distance greater than 24˚ is removed from the star pair catalog. The star pair with angular distance less than 0.16˚ (minimum required distance between stars 4.4μm) are removed from the star pair catalog [37]. This reduces the star pair catalog to 116,839-star pairs. The generation of star par ID is shown in Table 18.

$$cos\,\theta' = V_{ix}\,V_{jx} + V_{iy}\,V_{jy} + V_{iz}\,V_{jz} \quad (15)$$

Table 18: Star pair ID catalog

Star pair ID	i	j	cosϴ'
1	2337	2404	0.942008622
2	881	1692	0.942008792
--	--	--	--
116838	331	333	0.999995471
116839	2062	2063	0.999995871

The angular distance between the stars in the image is matched with $\cos\theta'$ in the catalog to identify the stars. The binary search technique is used for identifying stars but it consumes time of $2\log_2 N_p$, where N_p is the number of star pairs in the catalog. Hence we use K-Vector range search algorithm in which integer K is calculated from a linear equation fitting the angular distance between stars $\cos\theta'$. The K-Vector element represent the number of star pairs in catalog who's angular separation is equal to or just below the $\cos\theta'$. Equation 16 shows the

linear equation fitting. This star pair catalog is converted to K-Vector catalog and its shown in Table 19. The generation of K-Vector catalog and K-Vector search are detailed in the reference [38].

$$z(x) = mx + q = \cos\theta', \quad and \;\; k = int(x) \tag{16}$$

Table 19: K vector catalog

K(k)	i	j
0	2337	2404
2	881	1692
--	--	--
116837	331	333
116838	2062	2063

Chapter 4: Algorithm Development

The main objective of APST is to achieve arcsecond accuracy using commercially-off-the-shelf (COTS) components. The baffle, optics, image sensor, processor and other electronics of APST are selected from COTS. The COTS components have several advantages like low cost, high availability and less development time. But it's not customized to meet all the requirements of the system (e.g., the overall accuracy of the APST decreases due to aberration and distortion in the optics), hence there is a trade-off with the performance. This is compensated using a customized star selection algorithm for APST. The algorithm for APST contains four sections, which is shown in Figure 30. The conventional algorithm only uses image processing, identification, and determination. In APST, we implemented star selection method to improve the performance.

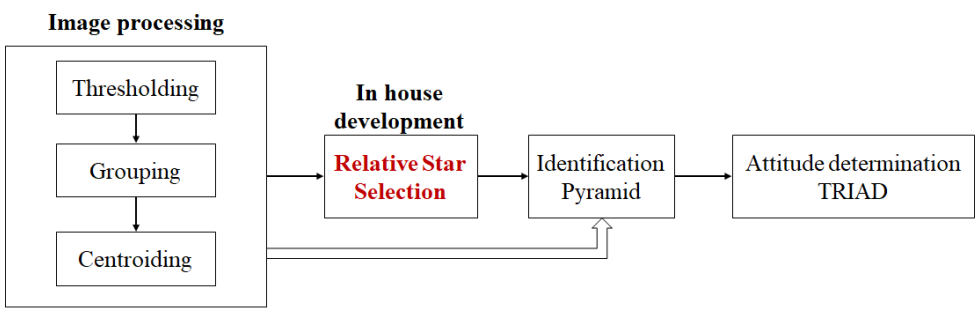

Figure 30: APST algorithm flow

In image processing and attitude determination, we follow the conventional methods. The image processing contains thresholding, labeling and weighted centroiding. The pyramid algorithm is used for star identification and triad is used for attitude determination. The relative star election algorithm is developed in-house and compared with the existing bright star selection

method. The star selection algorithm is not only to limited to nanosatellite platform, it can be used for star tracker operating in larger satellites.

4.1 Thresholding

The night sky image contains signals from stars and background noises from the image sensor (e.g., dark current, quantization noise, read noise). The thresholding extracts the signals from noise background. There are two types of thresholding global and local thresholding.

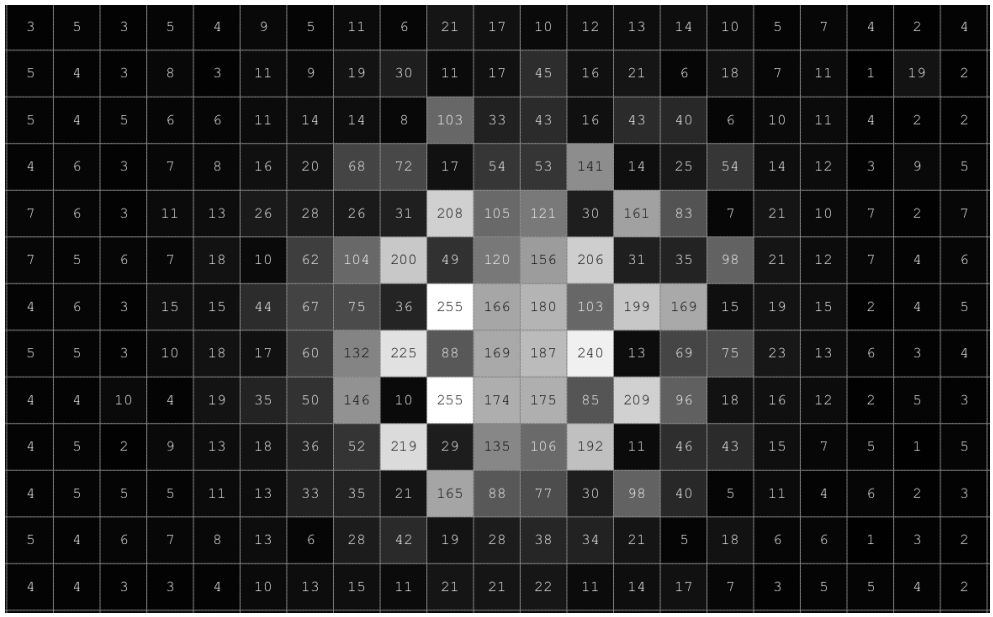

Figure 31: Star image before thresholding

The global thresholding used for uniform background illumination (e.g., night sky) and local thresholding used for nonuniform background illumination, which is the actual scenario in the orbit. Since we are testing uniform background night sky image, global thresholding meets our requirement. There are many types of filters, like mean, median, mode filters but in global

thresholding, we use 5σ for extracting signals. Figure 31 shows the image of a star before thresholding. The image contains background noise and foreground signal from a star. Every pixel in the image sensor has 0 to 255 value (8-bit). Total mean of all the pixel value in the image is calculated and 5σ is the threshold value of the image [39]. The threshold value of Cassiopeia and Lyra image is 12. The pixel intensity value below 12 is noise and above 12 are a signal from the star. Based on the threshold value, the pixels with intensity above 12 are extracted for further processing. We have tried to mean, 4σ, 5σ, 6σ filters. In comparison, 5σ extracts the signal accurately.

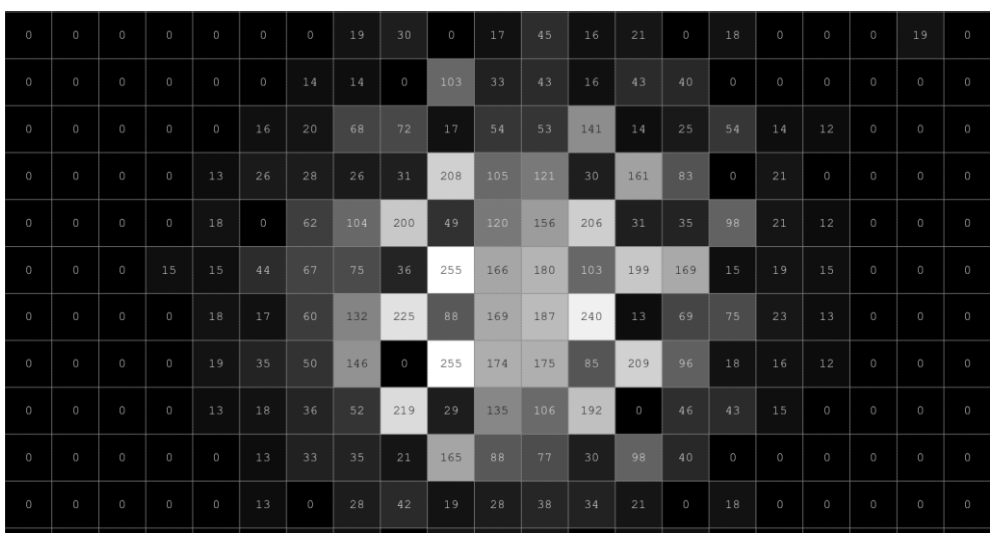

Figure 32: Star image after thresholding

4.2 Grouping

The optics in the star tracker is slightly defocused. Due to defocused optics, a star will spread over multiple pixels in the image sensor. Based on the optics and pixel size, we can determine the minimum and a maximum number of connected pixels required to identify it as a star. In APST, we consider a

minimum of 15 connected pixels and maximum of 250 connected pixels. If there are more than 250 connected pixels than the object is brighter than the apparent magnitude of zero. The sun, earth, earth's moon, other brighter planets are avoided using maximum connected pixels.

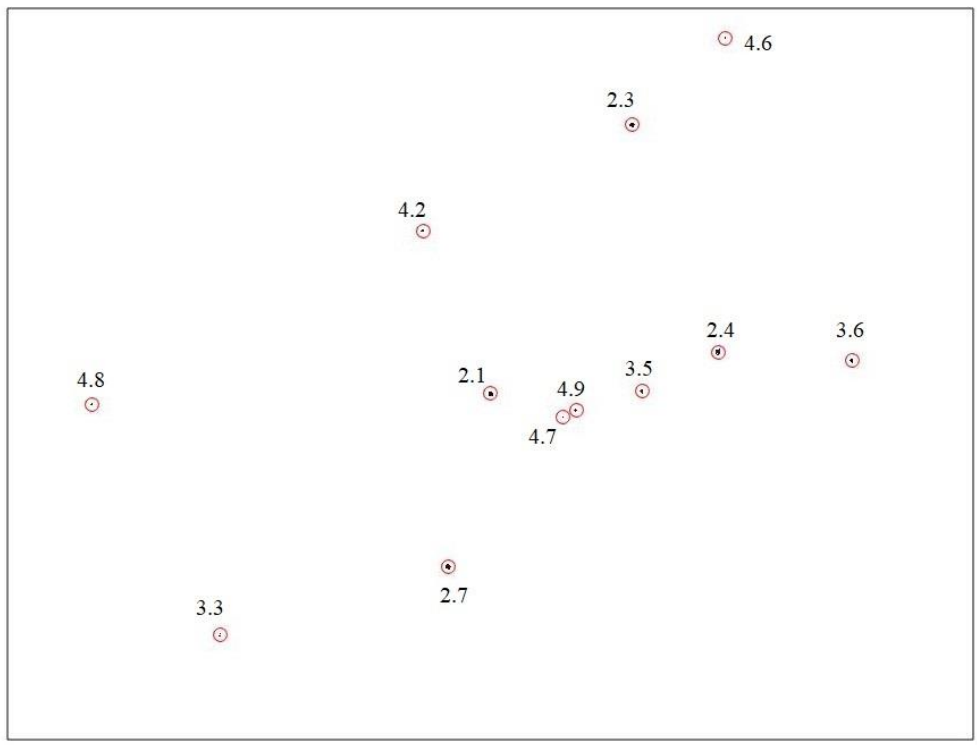

Figure 33: Detected stars in Cassiopeia

We use MATLAB function bwconncomp for labeling, extracted pixels searches it's neighboring pixels vertically, horizontally and diagonally (8-connectivity) to find it's connected pixels. A total number of 12 and 21 stars are detected in Cassiopeia and Lyra constellation respectively. The detected stars with an apparent magnitude in Cassiopeia and Lyra are shown in Figure 33 and 34. In Cassiopeia and constellation stars, up to 4.9 and 5.8 apparent

magnitudes are detected respectively. It's confirmed that APST will have 99% sky coverage (limiting magnitude: 5.5). The alternative options are grouping and labeling algorithms.

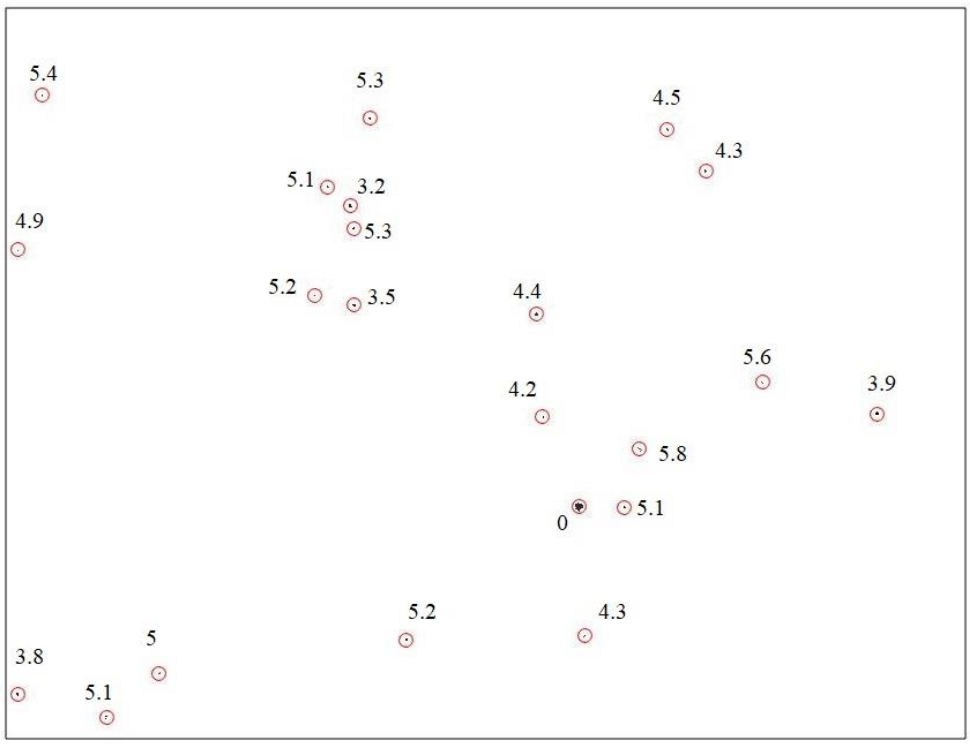

Figure 34: Detected stars in Lyra

Figure 35 shows the shape of the stars imaged by APST camera. On the left is the brightest of 2.1 magnitude and second is the 3.5 and third is 4.6 magnitude. The lower the magnitude is the brighter. The brighter star of 2.1 magnitude has better shape and pixel are not eroded in the middle. But in Cas 24 and 7 the shape is not regular and pixels in the middle are eroded. This is due to weaker signal and thresholding. We have to find a better way of thresholding for fainter stars. If the fainter stars are extracted in good shape, then centroiding will be more accurate. It

will improve the success rate of star identification. Hence dynamic thresholding which suits for a range of star brightness should be implemented in future.

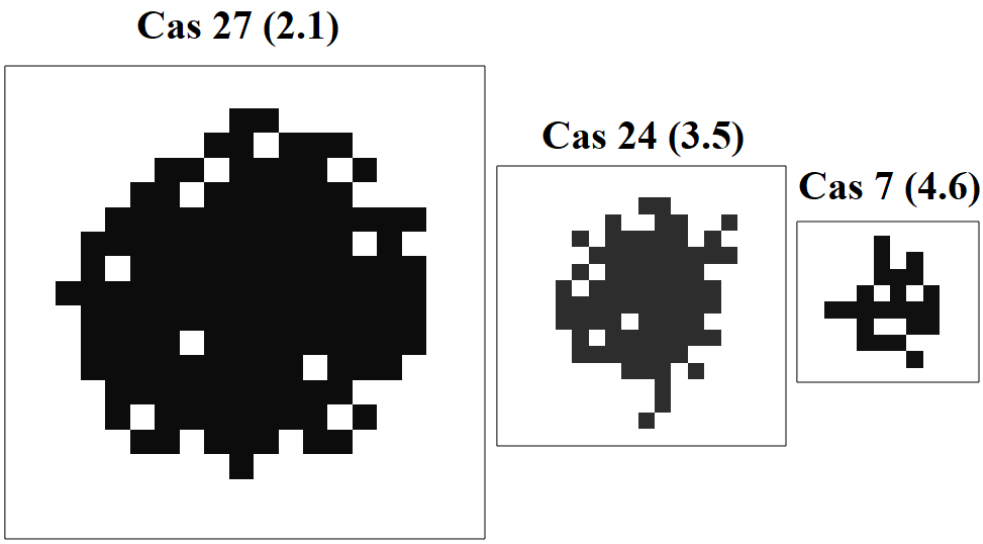

Figure 35: Image of stars in Cassiopeia based on brightness

4.3 Centroiding

There are two common types of centroiding algorithm, the center of gravity, and Gaussian curve fitting. The Gaussian curve fitting is more accurate but it's processor intensive and complex for pico star tracker. The center of gravity is less accurate but simpler and far less processor intensive. Hence, we use the center of gravity where weight is the brightness of pixels. Equations 17 and 18 are applied to each detected star to estimate the centroid of in x and y-axis. Cx is horizontal centroid position in pixels, Cy is vertical centroid position in pixels, n is a number of pixels in a star. Intensity is the brightness of the pixel ranging from 0 to 255. The centroid coordinates (Cx, Cy) of stars are converted to image sensor center frame of reference as x, y using Equation 19 and 20 (image sensor dimension:

2592x1944 pixels). The readers can use MATLAB function regionprop for weighted centroiding. The above-mentioned parameters of stars in Cassiopeia constellation is shown in Table 20 for reader's understanding. The centroid coordinates should be represented in 12 decimal points but due to limited space in table 3, we have mentioned in two decimal points. Figure 36 shows an image of a star Cas 27 with centroid location in pixels.

$$Cx = \sum_{i=0}^{n} \frac{[(pixel(i)\mathrm{x})\ \ (pixel(i)intensity)]}{(pixel(i)intensity)} \tag{17}$$

$$Cy = \sum_{i=0}^{n} \frac{[(pixel(i)\mathrm{y})\ \ (pixel(i)intensity)]}{(pixel(i)intensity)} \tag{18}$$

$$x = Cx - 1296 \tag{19}$$

$$y = Cy - 972 \tag{20}$$

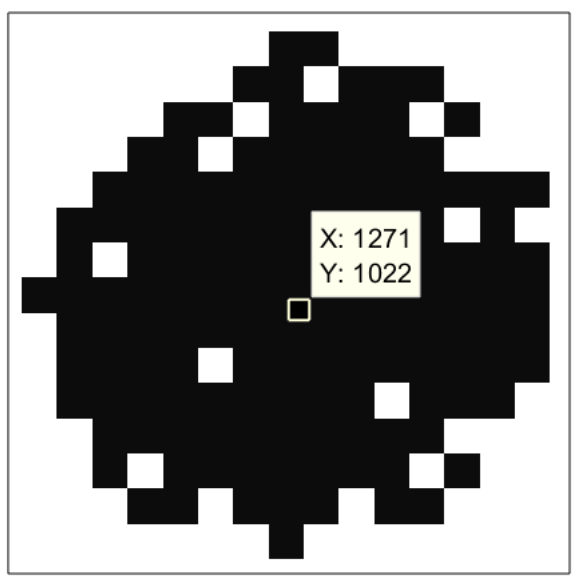

Figure 36: Weighted centroiding of star Cas 27

Table 20: Important properties of detected stars in Cassiopeia constellation

Name	HIP ID	Magnitude	No. of pixels	Centroiding coordinates	
				Cx (pixels)	Cy (pixels)
36 cas	6692	4.8	18	217.82	1050.00
45 cas	8886	3.3	39	555.16	1661.11
15 cas	2599	4.2	45	1092.27	592.81
37 cas	6686	2.7	146	1158.62	1481.10
27 cas	4427	2.1	141	1271.35	1022.43
28 cas	4422	4.7	16	1462.45	1084.43
26 cas	4292	4.9	25	1496.80	1065.75
11 cas	746	2.3	146	1646.22	312.09
24 cas	3821	3.5	78	1670.86	1015.76
18 cas	3179	2.4	175	1872.98	913.67
7 cas	117863	4.6	23	1890.66	81.80
17 cas	2920	3.6	45	2225.97	935.71

4.4 Angular Distance Measurement Error Analysis

Once the centroid of all the stars in the image is determined. Then the angular distance between stars is calculated using Equation 21. The centroiding coordinates two stars (x_1, y_1), (x_2, y_2) and focal length (f) in pixels are required to measure angular distance between stars in $\cos\theta$" [40]. In order to evaluate the measurement error of APST the actual angular distance between stars from catalog $\cos\theta'$ and measured angular distance from image $\cos\theta''$ is compared using Equation 22. The two constellation Cassiopeia and Lyra are imaged using APST hardware and the measurement error is estimated in $\cos\theta''$ and arcsecond, its shown in Table 21 and 22 respectively.

$$\cos\theta'' = \frac{x_1\,x_2 + y_1\,y_2 + f^2}{\sqrt{x_1^2 + y_1^2 + f^2}\sqrt{x_2^2 + y_2^2 + f^2}} \tag{21}$$

$$Measurement\ \ error = |\cos\theta' - \ \cos\theta''| \tag{22}$$

There are 66 and 153-star pairs in Cassiopeia and Lyra image respectively, the

measurement error for all the star pairs are estimated using Equation 22. In Table 21 the minimum measurement error of 1.46 E-07 and maximum of 6.02 E-05 $(\cos\theta)$ is given. In Table 22 the minimum measurement error of 1 and maximum of 264 arcsecond is given. In order to successfully identify four stars using pyramid algorithm the measurement error should be with in a limit otherwise the stars will be misidentified or unidentified. Based on the literature review and data analysis, we estimated that if error is less than E-06 $(\cos\theta)$ or 30 arcsecond then stars will be successfully identified using pyramid algorithm.

Table 21: APST angular distance measurement error in $\cos\theta$

Measurement error	Cassiopeia $(\cos\theta)$	Lyra $(\cos\theta)$
Minimum	1.46 E-07	2.08 E-07
Maximum	6.02 E-05	3.42E-04
Mean	1.04 E-05	9.66 E-05

Table 22: Angular distance measurement error in arcsecond

Measurement error	Cassiopeia (arcsecond)	Lyra (arcsecond)
Minimum	1	1
Maximum	49	264
Mean	17	115

The angular distance measurement error of APST is above the margin in Table 21 and 22. There are many factors involved in the increase in measurement error. They are a low signal, image sensor noise, lens aberration, lens distortion, inaccurate thresholding, and centroiding. After analyzing the image and data from APST, we estimated the main two contributions of error is lens distortion and centroiding. In APST, we used from lens from COTS hence it has significant distortion which cannot be avoided. Currently, we don't have test setup to measure

the distortion of the lens exactly, hence distortion correction algorithm is not implemented. This will be one of the future work for APST. The second factor is centroiding because the bright stars have low measurement error and faint stars has high measurement error. But the dominant factor is a distortion of the lens. The distortion graph of B3M16018 lens used in APST is shown in Figure 37. It has maximum radial distortion of - 0.65%. There are three types of distortion parallel, pincushion and mustache distortion. The B3M16018 has pincushion distortion, hence it's given in negative.

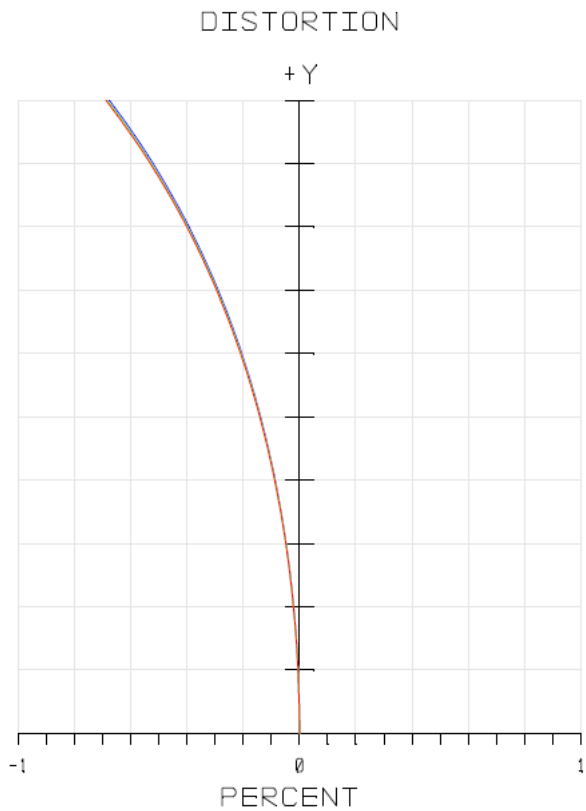

Figure 37: Radial distortion of B3M16018 lens used in APST

The distortion graph in Figure 37 doesn't show the details of the distortion

value radially. The distortion estimation model shown in Equation 23 and 24 is used to determine radial distortion of APST [41]. The u_d, v_d are distorted point (centroid of star in x and y). The u_o, v_o are the optical center of the image sensor. The $k_{1max} \leq (\frac{2}{r^2})$ is the distortion coefficient and u, v are undistorted location point of star, $\alpha = \beta = \frac{pixel\ \ size}{focal\ \ length}$, estimated $k_{1max} = 0.12$.

$$u = u_d + (u_d - u_0)k_1\left[\frac{(u_d-u_0)^2}{\alpha^2} + \frac{(v_d-v_0)^2}{\beta^2}\right] \quad (23)$$

$$v = v_d + (v_d - v_0)k_1\left[\frac{(u_d-u_0)^2}{\alpha^2} + \frac{(v_d-v_0)^2}{\beta^2}\right] \quad (24)$$

$$r = pixel\ \ size \times \frac{image\ \ sensor\ \ lenght}{2} \times \sqrt{2} \quad (25)$$

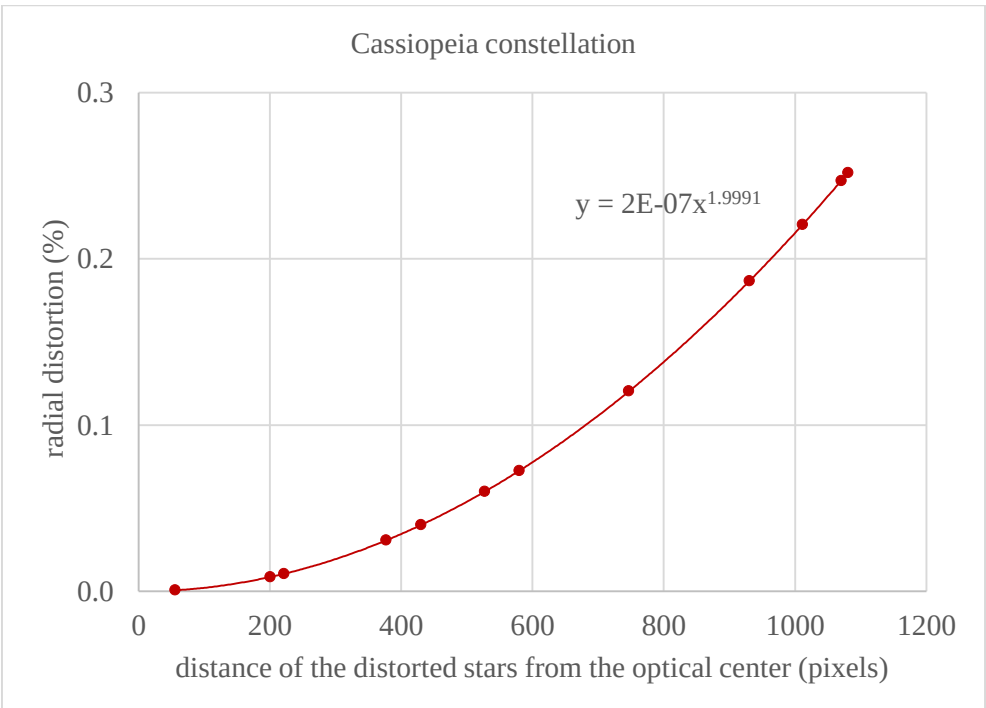

Figure 38: Radial distortion of APST in Cassiopeia image

The radial distortion of the APST is estimated using Equation 2 is shown in Figure 38 and 39. In Equation 26 Distorted Distance (DD) and Undistorted Distance

(UDD) are used to estimate the radial distortion. Figure 38 and 39 clearly shows that as stars move radially outward, the distortion increases exponentially. The maximum distortion of 0.5% is estimated.

$$Radial\ \ distortion\ \ \% = \left(\frac{DD-UDD}{UDD}\right) \times 100 \qquad (26)$$

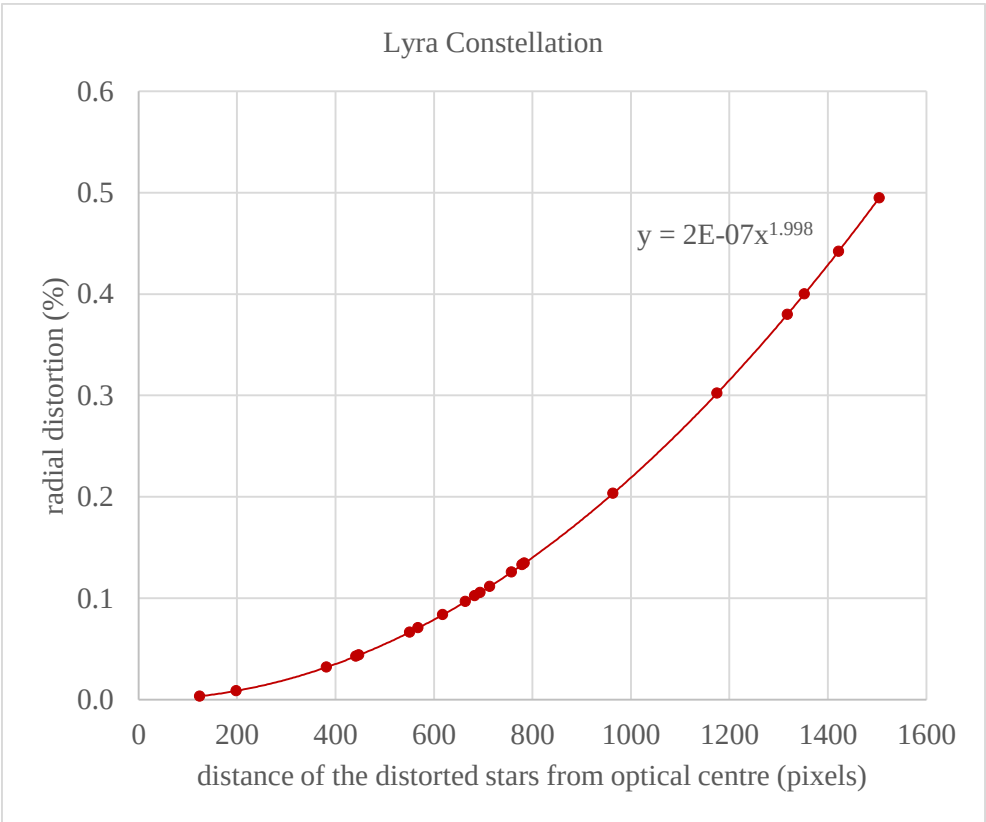

Figure 39: Radial distortion of APST in Lyra image

In the above distortion model, let's consider two points in 1-dimensional plane r_1, r_2 are actual distance and r_{1m} and r_{2m} are measured distance along the radial distance (r) from the optical center and distortion is function of radial distance $f(r_1)$, $f(r_2)$, ε is the distance between r_1 and r_2 .

Consider point r_1 greater than r_2

$$f(r_1) > f(r_2) \quad (27)$$

$$f(r_1) = f(r_2) + \varepsilon \quad (28)$$

$$r_{1m} = r_1 + f(r_1) \quad (29)$$

substitute Equation 28 in 29

$$r_{1m} = r_1 + f(r_2) + \varepsilon \quad (30)$$

$$r_{2m} = r_2 + f(r_2) \quad (31)$$

$$Actual\ \ distacne = |r_1 - r_2| \quad (32)$$

$$Measured\ \ distance = |r_{1m} - r_{2m}| \quad (33)$$

$$Measrued\ \ distance = r_1 + f(r_2) + \varepsilon - r_2 - f(r_2)$$

$$Measured\ \ distance = r_1 - r_2 + \varepsilon \quad (34)$$

$$Error = Actual\ \ distance - Measured\ \ distance \quad (35)$$

$$Error = r_1 - r_2 - r_1 + r_2 - \varepsilon$$

$$Error = -\varepsilon$$

The above derivation shows that the distortion is a function of the distance between two points in the 1-dimensional plane. The measurement error between two stars depends on the distance between that two stars. If the angular distance between two stars (ε) is larger than the distortion is higher which leads to higher

measurement error and vice versa. In order to validate numerically, we divided the image sensor frame into five parts form optical center as R1, R2, R3, R4, R5. The pixels from 0 (optic center) to 250 is R1, 0 to 500 is R2, 0 to 750 is R3, 0 to 1000 is R4, 0 to 1250 is R5. We made five cases for our analysis.

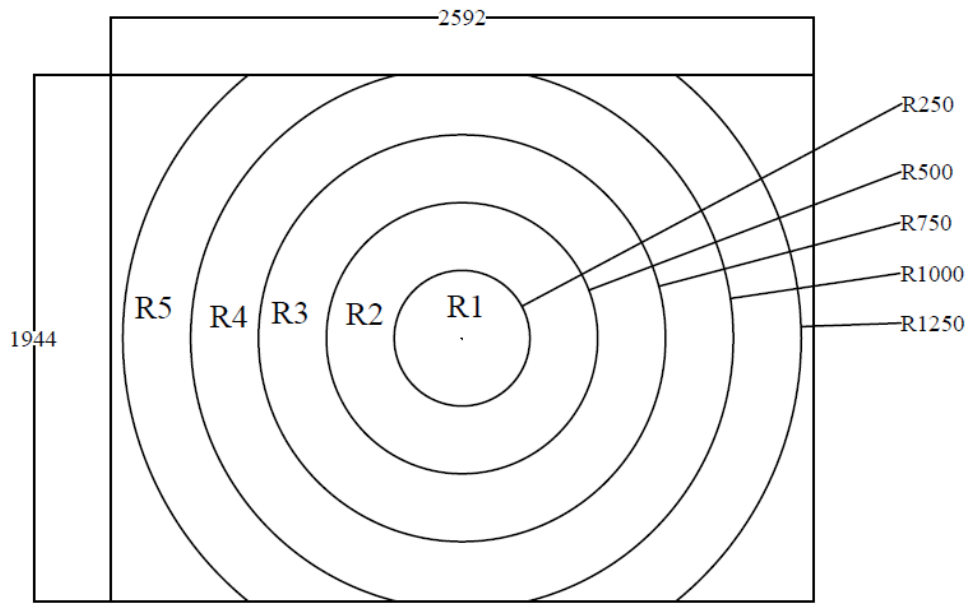

Figure 40: Division of image sensor from optic center

In case1 two stars within R1is considered, which has minimum inter-distance of 39. The angular measurement error is 1.78 E-07 shown in Table 23. The R1 with minimum distance has lowest measurement error. In Table 24 two stars in R1 with a maximum distance of 201 pixels has measurement error 3.42 E-06. The error has increased considerably due to increase in distance from 39 to 201 pixels. In Table 25, R5 is considered. R5 has maximum distortion because it's almost at the edge of the image. But we considered two stars with a minimum distance of 698 in R5 and it has measurement error 5.6E-06 and two stars with a maximum distance of 1250

in R5 has a measurement error of 2.09 E-05

Table 23: Case 1 (R1 minimum)

Star No.	Star location	Distance between two star (pixels)	Angular distance measurement error ($\cos\theta$)
1	R1	39	1.78E-07
2	R1		

Table 24: Case 2 (R1 maximum)

Star No.	Star location	Distance between two star (pixels)	Angular distance measurement error ($\cos\theta$)
1	R1	201	3.42E-06
2	R1		

Table 25: Case 3 (R5 minimum)

Star No.	Star location	Distance between two star (pixels)	Angular distance measurement error ($\cos\theta$)
1	R5	698	5.60E-06
2	R5		

Table 26: Case 4 (R5 maximum)

Star No.	Star location	Distance between two star (pixels)	Angular distance measurement error ($\cos\theta$)
1	R5	1250	2.09E-05
2	R5		

Table 27: Case 5 (R1-R5)

Star No.	Star location	Distance between two star (pixels)	Angular distance measurement error ($\cos\theta$)
1	R1	1054	1.06E-05
2	R5		

In Table 27 we considered a star in R1 and R5, it has a distance of 1054 pixels and measurement error of 1.06 E-05. Since maximum distortion of lens is 0.65 %, if the location of the stars is near to the optic center or edge of the image is not the crucial factor. The crucial factor is the distance between two stars. Even if two stars are located at the edge of the image but with minimum inter-distance the measurement error is lower. At the same time if one of the stars located in the optic center and the other one near the edge of the image then inter-distance is high which leads to maximum measurement error.

4.5 Star Selection

In last section, we discussed the measurement error in APST. The pyramid algorithm is used for star identification since the measurement error is high, most of the stars are misidentified or unidentified. In general, the night sky image contains up to 30 stars. But the pyramid algorithm only needs four stars for identification. In general, four-star are selected based on brightness or stars from the center of FOV because brighter stars will have accurate centroiding and stars from the center of FOV will have less distortion due to this factors the measurement error will be less. But the above cases are not effective in selecting stars with less measurement error for APST, hence we developed novel star selection method known as relative star selection for APST.

4.5.1 Relative Star Selection

Based on the analysis on angular distance measurement errors of stars in Cassiopeia and Lyra image in the last section, we identified that if the inter-distance between two stars is short, then the angular distance measurement error of

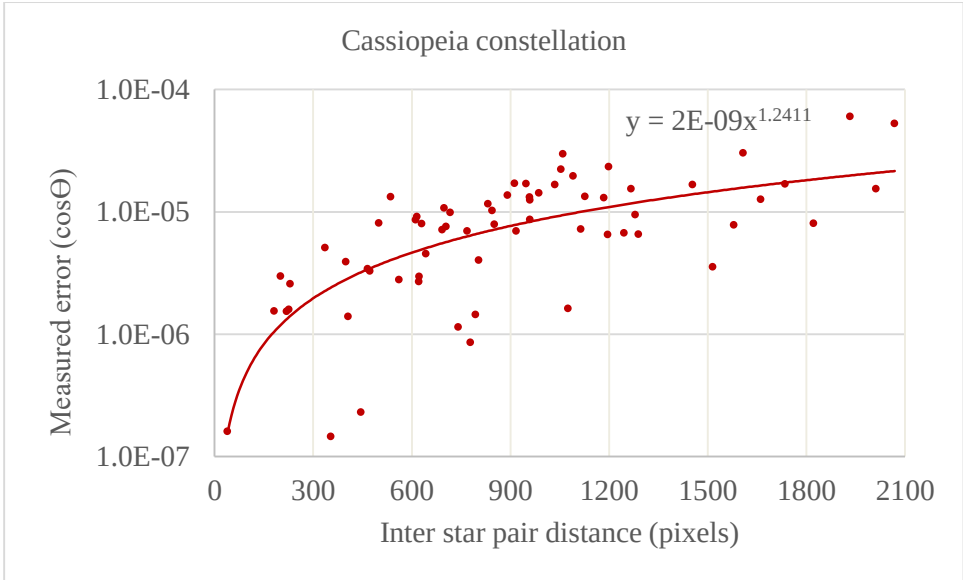

Figure 41: Measured error distribution over distance between stars

the star pair is low. Based on this fact, the distance between stars is measured in a 2d plane in pixels using Equation 27 and sorted the distance in ascending order in the x-axis and its corresponding measurement error is plotted in the y-axis. This graph in Figure 40 and 41 shows the error distribution in Cassiopeia and Lyra image respectively. Figure 40 and 41 shows that as the distance between the stars increases the measurement error increases exponentially. Based on the APST measurement error analysis, the star pairs below E-05 can be successfully identified.

$$Inter\ \ star\ \ pair\ \ distance = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} \quad (27)$$

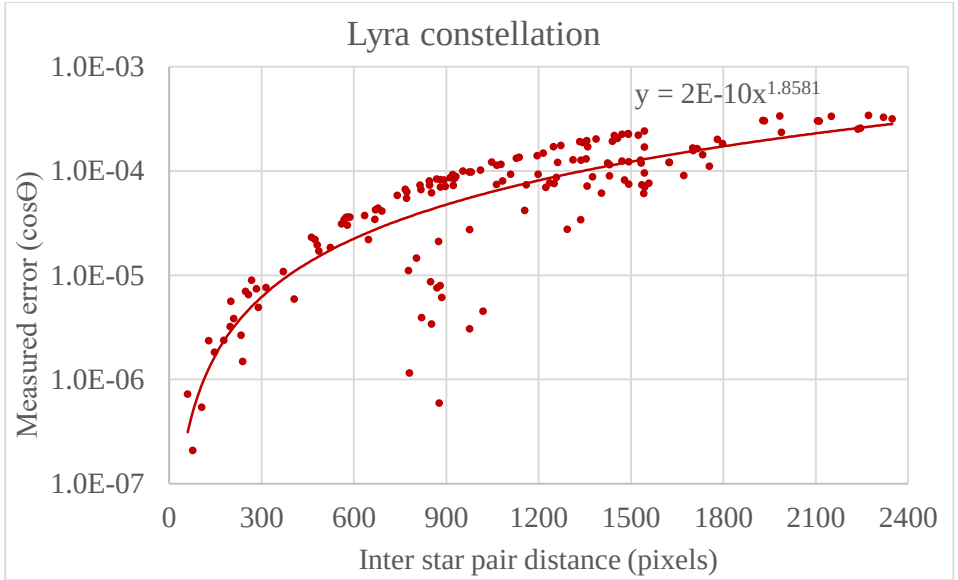

Figure 42: Measured error distribution over distance between stars

In Figure 41 and 42, the first 31 and17 pairs have the measurement less than E-05 respectively. On further analysis, we estimated that in order to successfully identify three stars, we need only first 15 star pairs. The first 15 inter star pair distance and its corresponding star pair number, a measurement error of Cassiopeia and Lyra is shown in Table 23. But the measurement error of each star tracker varies based on its image sensor, lens, and centroiding error based on that number of star pairs should be decided.

Among the 15 star pairs (30 stars) four stars have to be selected with pyramid pattern for successful identification. Four stars with pyramid pattern are selected among the 15-star pairs in Table 23. In Cassiopeia constellation star pairs (6,7), (7,9) and (5, 7) has the common value of 7, it forms a pyramid, hence star number (6,7, 9, 5) are selected, it's shown in Figure 42. The other possible selection in Cassiopeia constellation are (3,5,7,9) and (4,5,6,7). In Lyra constellation star pairs

(8,10), (7,8) and (8,11) has the common value of 8, it forms a pyramid, hence star number (7,8,10,11) are selected.

Table 28: First 15 star pairs and its corresponding measurement error

Cassiopeia constellation			Cassiopeia constellation		
Inter star pair distance (pixels)	Star pair Number	Measurement error (Cos Θ)	Inter star pair distance (pixels)	Star pair number	Measurement error (Cos Θ)
39	6, 7	1.6E-07	60	8, 10	7.2E-07
181	7, 9	1.5E-06	77	7, 8	2.1E-07
201	5, 6	3.0E-06	106	6, 9	5.4E-07
219	6, 9	1.5E-06	129	15, 17	2.4E-06
226	9, 10	1.6E-06	148	7, 10	1.8E-06
230	5, 7	2.6E-06	178	19, 20	2.4E-06
336	8, 11	5.1E-06	199	17, 18	3.2E-06
354	10, 12	1.5E-07	201	4, 5	5.6E-06
400	5, 9	3.9E-06	210	9, 10	3.8E-06
406	7, 10	1.4E-06	234	6, 10	2.6E-06
445	6, 10	2.3E-07	239	7, 11	1.5E-06
465	3, 5	3.4E-06	249	15, 18	7.0E-06
472	4, 5	3.3E-06	258	8, 11	6.5E-06
500	4, 6	8.1E-06	268	1, 4	8.9E-06
536	4,7	1.3E-05	284	6, 8	7.4E-06

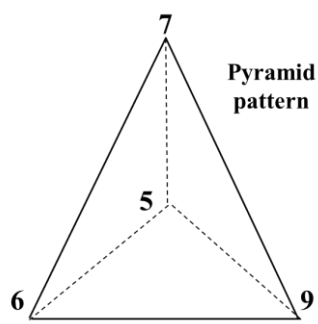

Figure 43: Pyramid star pattern

Using the above method, the four stars are selected for each image. Since the measurement error of this three-star pairs is below 9.0E-06, all the three stars will be identified successfully in the first iteration of the identification algorithm. Hence the proposed the star selection method guarantees successful identification of four

stars in less time for APST. We name this as relative selection algorithm. The concept and execution of relative star selection algorithm are explained with examples in above. The flow of the relative star selection algorithm is summarized briefly in step by step in the following Figure 43.

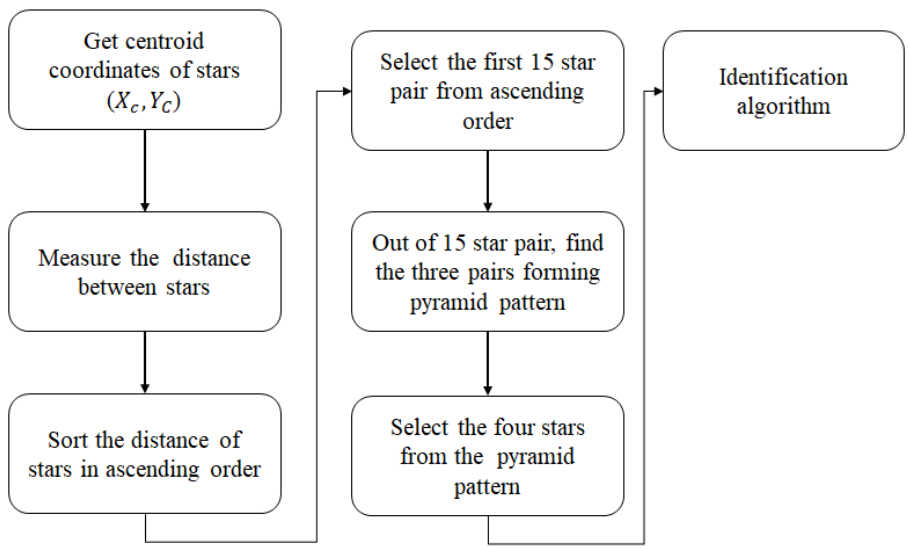

Figure 44: Relative star selection algorithm

The relative star selection algorithm is applied to night sky image of Cassiopeia and Lyra show in Figure 44 and 45 respectively. In Figure 44, there are 12 stars in total, out of 12 s 4 stars are selected using relative star selection algorithm which is shown inside the rectangular box. It's easy to understand from the image that these four stars are located closer to each other. As we explained earlier that relative star selection algorithm selects the four stars which are located closer to each other because the closer the star pair lower the measurement error.

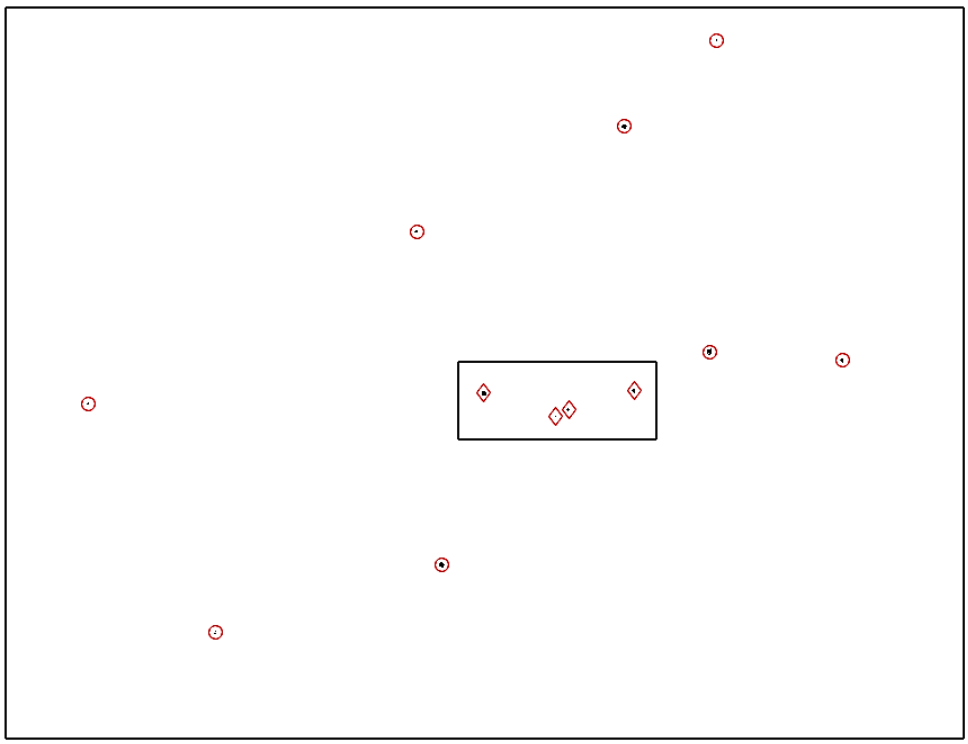

Figure 45: Stars in rectangular box are selected using relative star selection method

In Figure 45, there are 22 stars in total. Out of 22 stars, four stars are selected using relative star selection method. These four stars are located near to each other and shown in a rectangular box. One of the advantages of the relative star selection method is irrespective of a number of stars in the image, it selects only four stars hence time required for identification of star is less. The selected four stars from Cassiopeia and Lyra image are used in pyramid algorithm for star identification and in the following attitude is determined using TRIAD. The Pyramid and triad algorithm will be explained in the following sections. Using star simulator 75-star constellation are tested with relative star selection algorithm and results are discussed in chapter 5.

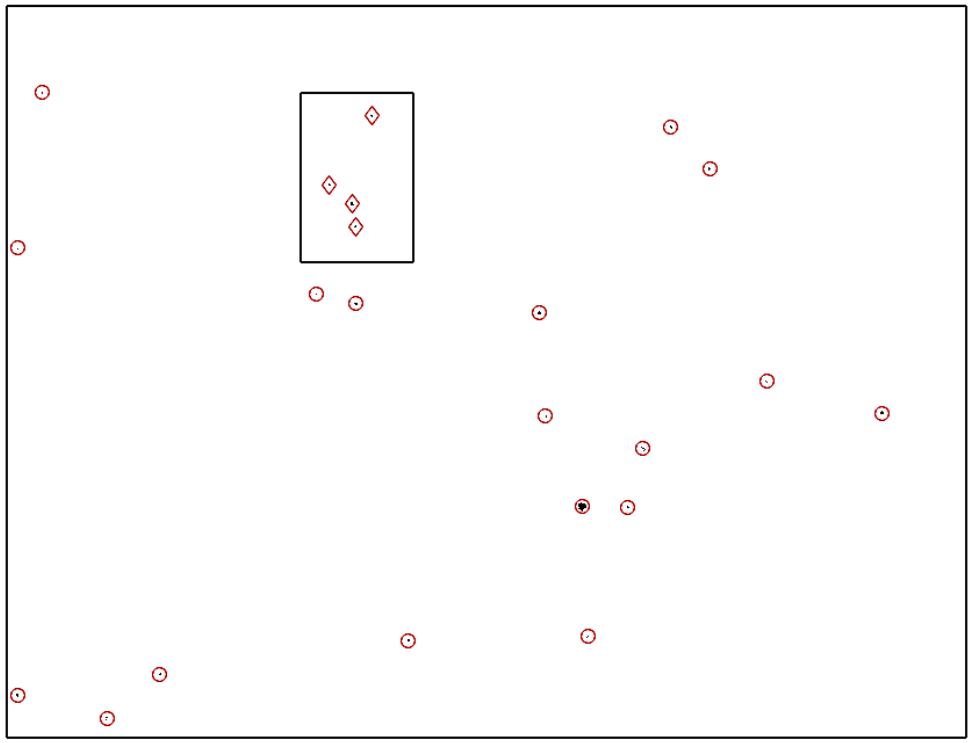

Figure 46: Stars in rectangular box are selected using relative star selection method

The relative selection algorithm will be very effective around equatorial region, in particular from -60° to 60 latitudes because the star distribution is denser around the galactic plane and sparse around the galactic pole. When the boresight of the APST is around galactic plane more than 10 stars in the FOV is typical but around galactic poles, many of the FOV see only 3 stars. The Fig.6 shows the star distribution over the sky, the latitude and longitude are divided by 15° and 20° respectively, which visualize the FOV of APST. The relative selection algorithm will perform successfully around the equatorial region in less time. In addition, this relative selection algorithm will be advantageous for the star tracker with higher limiting magnitude because the number of stars in sky increases exponentially as the magnitude of star increases. Based on the literature review, the star distribution

over the entire sky, galactic plane, and galactic poles for various apparent star magnitude are shown in Table 29 [22].

Table 29: Average star density

Apparent star Magnitude	Average Star Density (/deg²)		
	Entire sky	Galactic plane	Galactic pole
5	0.05	0.08	0.04
6	0.15	0.32	0.13
7	0.47	1.32	0.35
8	1.53	4.51	0.88

4.5.2 Bright Star Selection

The bright star selection is a conventional method used for selecting stars for star trackers. In this method the stars are prioritized based on brightness because the bright stars have accurate centroiding compared to the faint stars, hence the attitude is determined with higher accuracy. The bright star selection method is implemented to compare the performance of relative star selection. In bright star selection method, stars in the image are sorted in descending order based on a number of pixels it contains. Second, the four possible combinations of all the stars is obtained. Third, then each four-star combination is selected in serial for star identification. The combinations are processed one by one until the solution is found. Figure 47 shows the number of combinations of the corresponding number of stars in the image. The graph shows that as the number of stars in the image increases the number of combination increases exponentially.

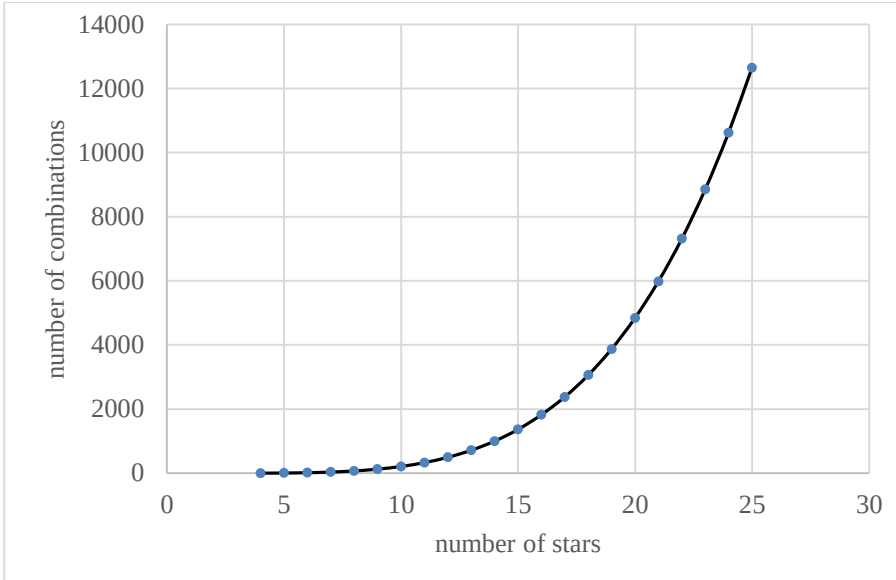

Figure 47: Number of combinations for the number of stars in the image

The bright star selection applied to Cassiopeia constellation. There are 26 stars in total detected in the image. These 26 stars are sorted in descending order based on a number of pixels the star contains. Then out of 26 stars, 4-star subset combination is formed. Based on the Equation 28, c is combinations, n is total number of star and r is sample. The n and r is 26 and 4 respectively which forms 14,950 possible combinations. These 14,950 combinations are processed one by one until the four stars are identified using pyramid algorithm. In Cassiopeia image, four stars are identified at 25th combinations. In Figure 48 the four stars in the square box are selected using bright star selection algorithm.

$$c(n,r) = \frac{n!}{(r!(n-r)!)} \tag{28}$$

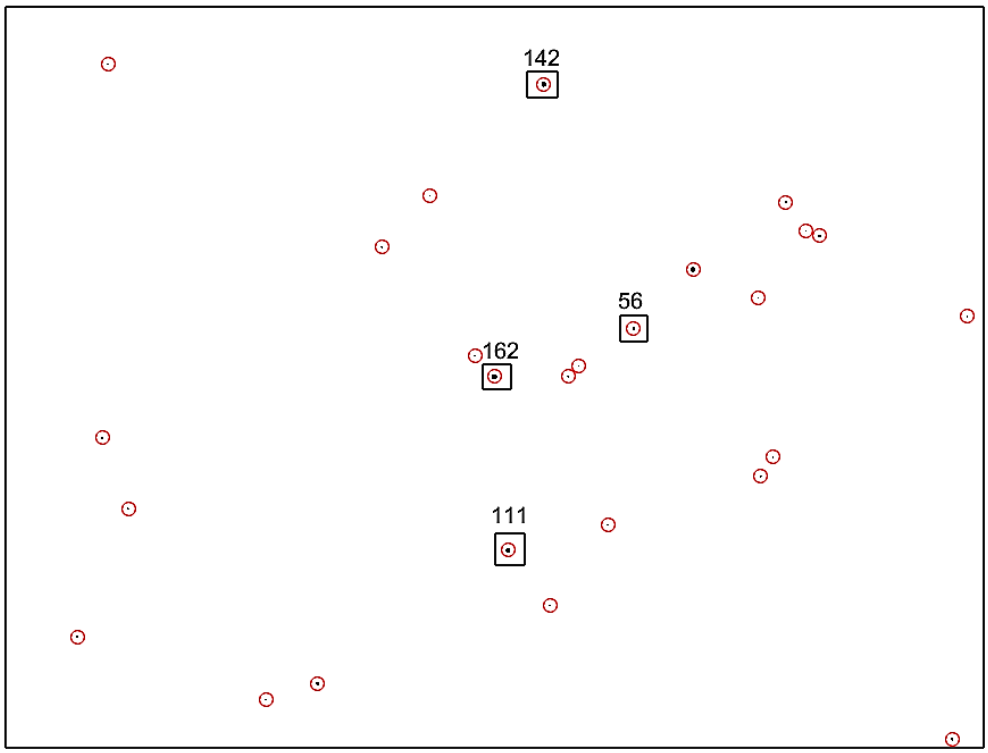

Figure 48: Stars in box are selected based on number of pixels (Cassiopeia)

The four stars selected using bright star selection contains a higher number of pixels in relative to other stars in the image. Figure 48 shows the number of pixels selected stars contains. The four stars are identified on the 25th combination, which means first 24 combinations cannot able to identify the stars because in bright star selection the stars are selected based on the brightness which means four stars can be distant apart. As we know when inter-distance between stars is longer then the measurement error is high which leads to unidentified or misidentified. Also as the number of combination increases the processing time for identification also increases, hence bright star selection is time-consuming and less successful when compared to relative star selection method. The bright star selection method is implemented in Lyra constellation. The same procedure is followed. The four stars

are identified at the 2nd combination. The selected are shown inside a square. The accuracy, success rate, processing time of bright star selection are discussed in detail in chapter 5.

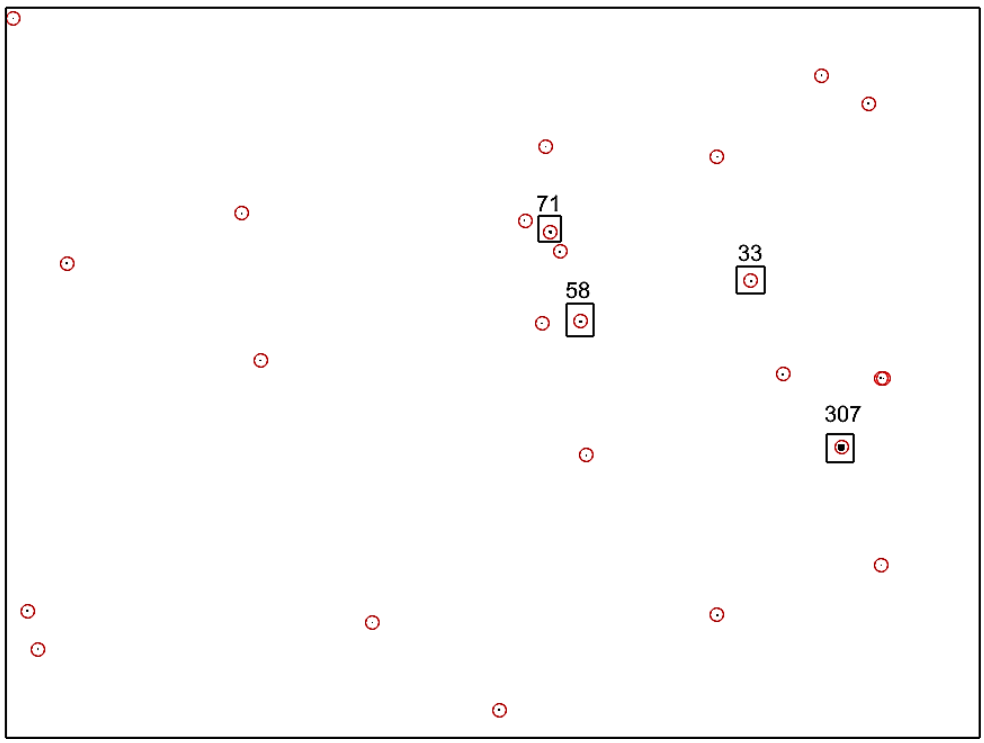

Figure 49: Stars in box are selected based on number of pixels (Lyra)

4.6 Identification

In identifying stars, we use the sub-graph which uses the angular distance between stars. There are many methods for identifying stars, the four-star matching algorithm is more robust, successful and faster. The four-star matching algorithm also known as pyramid algorithm [42]. In order to successfully identify stars, there should be minimum four stars in the image.

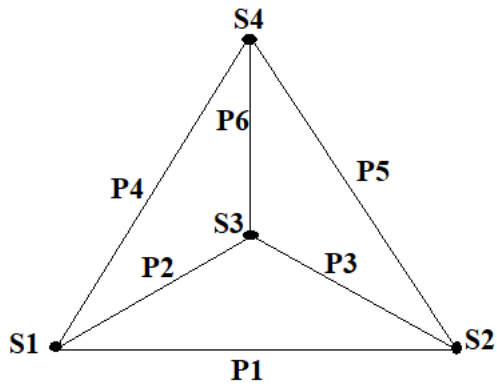

Figure 50: Four star identification

Figure 50 shows the model of the four-star image. The four stars are represented as S1, S2, S3, S4 and these four stars forms 6 pairs. The star S1, S2 forms first pair P1 and so on. The P1 is the angular distance between S1 and S2. Once there are four stars in the image angular distance of 6 star pairs are calculated. In the onboard catalog, the angular distance between the stars is calculated and ordered which is known as K-Vector catalog. In Equation 29 and 30, p represents the angular distance between each star pair. e is the star sensor accuracy or error, based on analysis error (e) is chosen to be E-06. If e is too small. It will exclude the actual star pair ID and if it's too large there is the possibility of a mismatch because there are many star pairs with similar angular distance. The floor and ceil are rounding functions. The possible star pairs ID is retrieved from K-Vector catalog between K top and bottom are retrieved using Equation 29 and 30.

$$Kb = floor\left(\frac{((p+e)-q))}{m}\right) \quad (29)$$

$$Kt = Ceil\left(\frac{((p-e)-q))}{m}\right) \quad (30)$$

For each star pair P1, P2, P3, P4, P5, P6, the k-vector filters the 30 possible stair

pairs out of 116,840 stair pairs in K-Vector catalog. Out of these 180star pairs, 4-star ID has to be identified. At first reference star ID has to be identified, we look for common star id in at least three pairs. Once the reference star is found, it's easier to find other three stars in the corresponding pair. This method of identification is known as pivot star identification. Table 30 shows the admissible pairs for star pair P1, P2, and P3 of Cassiopeia image. Among the admissible star pairs P1, P2, and P3, the reference star ID 1054 is identified. The star ID of 1049, 1042, 1077 located in the corresponding column of 1054 is the other three stars. Hence using pivot star identification all the four stars 1054, 1049, 1042, 1077 is identified.

Table 30: Pivot star Identification

Admissible pairs of P1		Admissible pairs of P2		Admissible pairs of P4	
1116	1124	1942	1959	408	425
2453	2455	1823	1872	2563	2579
1003	1016	2501	2504	529	541
2176	2179	2447	2456	963	973
1007	1016	116	119	**1054**	**1077**
1049	**1054**	2366	2374	1776	1787
329	333	**1042**	**1054**	254	267
196	199	889	912	1188	1198
819	823	272	285	2003	2019
687	688	429	437	1131	1144

4.7 Attitude Determination

After the star identification, the next step is the attitude determination. The attitude is determined using the TRIAD algorithm. Out of four stars, TRIAD needs only two stars which are not parallel for attitude determination. The coordinates of

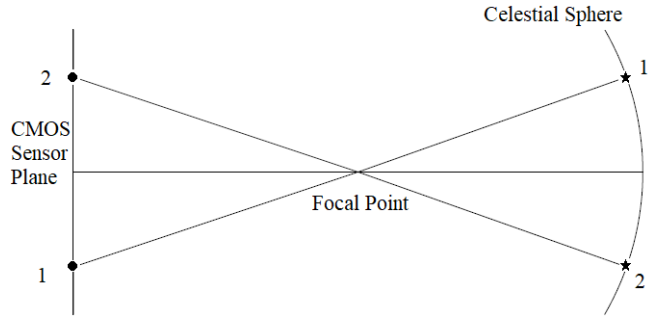

Figure 51: Celestial sphere projection in pinhole camera

two stars (V) obtained from mission catalog matched with the same two stars in the body frame (W). The rotation matrix (R) is obtained by transforming from ICRS frame to body frame using the Equation 31 and 32. The Equation 31 and 32 is overdetermined, hence a solution required construction of two tiads $(\hat{C}_i\ \hat{S}_i)$ $i = 1, 2, 3$

$$R\hat{V}_1 = \hat{W}_1 \tag{31}$$

$$R\hat{V}_2 = \hat{W}_2 \tag{32}$$

$$\hat{r}_1 = \hat{V}_1 \qquad \hat{r}_2 = \frac{\hat{V}_1 \times \hat{V}_2}{|\hat{V}_1 \times \hat{V}_2|} \qquad \hat{r}_3 = \frac{\hat{V}_1 \times (\hat{V}_1 \times \hat{V}_2)}{|\hat{V}_1 \times \hat{V}_2|} \tag{33}$$

$$\hat{s}_1 = \hat{W}_1 \qquad \hat{r}_2 = \frac{\hat{W}_1 \times \hat{W}_2}{|\hat{W}_1 \times \hat{W}_2|} \qquad \hat{r}_3 = \frac{\hat{W}_1 \times (\hat{W}_1 \times \hat{W}_2)}{|\hat{W}_1 \times \hat{W}_2|} \tag{34}$$

$$R\hat{r}_i = \hat{s}_i \tag{35}$$

$$R = [\hat{s}_1\ \ \hat{s}_2\ \ \hat{s}_3]\ [\hat{r}_1\ \ \hat{r}_2\ \ \hat{r}_3]^T \tag{36}$$

The rotation matrix (R) can be transformed to quaternion and Euler angles. The TRIAD is deterministic approach hence it's faster but when measured star vector is wrong, the resulting attitude is incorrect. The QUEST provides an optimal solution with as fast as TRIAD.

Chapter 5: Simulation Results

The star simulator is developed based on the results from the night sky image. In star simulator HIPPARCOS catalog is used to simulate the location (right ascension and declination) of stars and the ICRS coordinates is converted to sensor coordinates. The noise model is implemented based on the night sky testing result, it contains dark current noise from the sensor using Gaussian fit and lens distortion noise from the lens using exponential fit and size of a pixel is estimated using fourth order polynomial model.

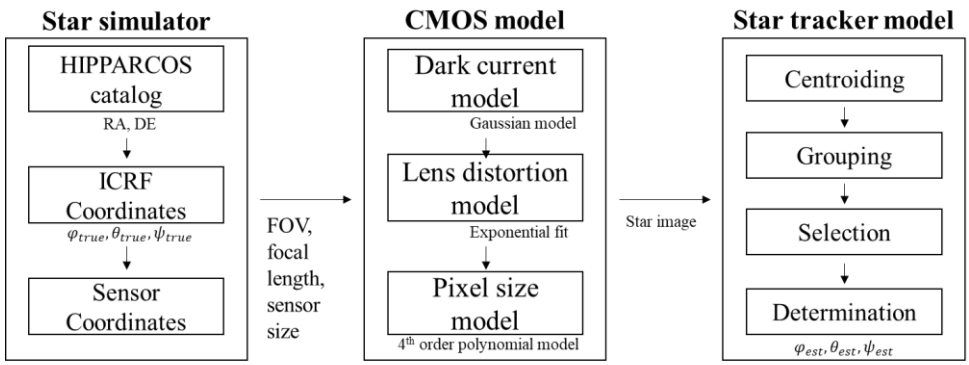

Figure 52: Star tracker Simulator of APST

The star tracker model contains thresholding, labeling, centroiding, star selection, identification and attitude determination. Using the star simulator, the performance of relative and bright star selection is analyzed. In total 75-star constellation are tested in a simulator, accuracy in pitch, yaw, roll and the processing time of the relative and bright star algorithm is measured. The details of the 75-star constellation, identified star HIP ID, and their respective, right ascension, declination and rotation are given in Appendix C.

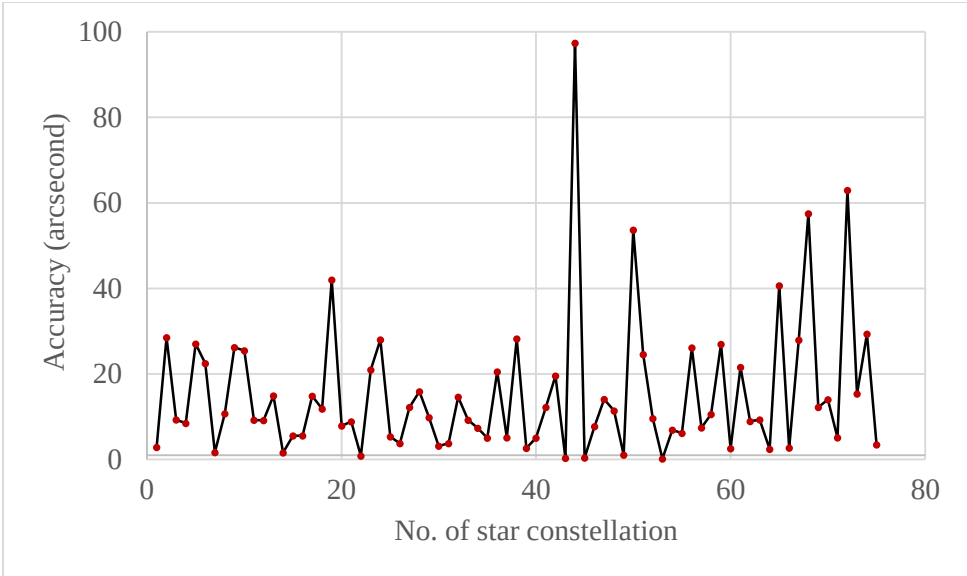

Figure 53: Relative Star Selection accuracy in pitch axis

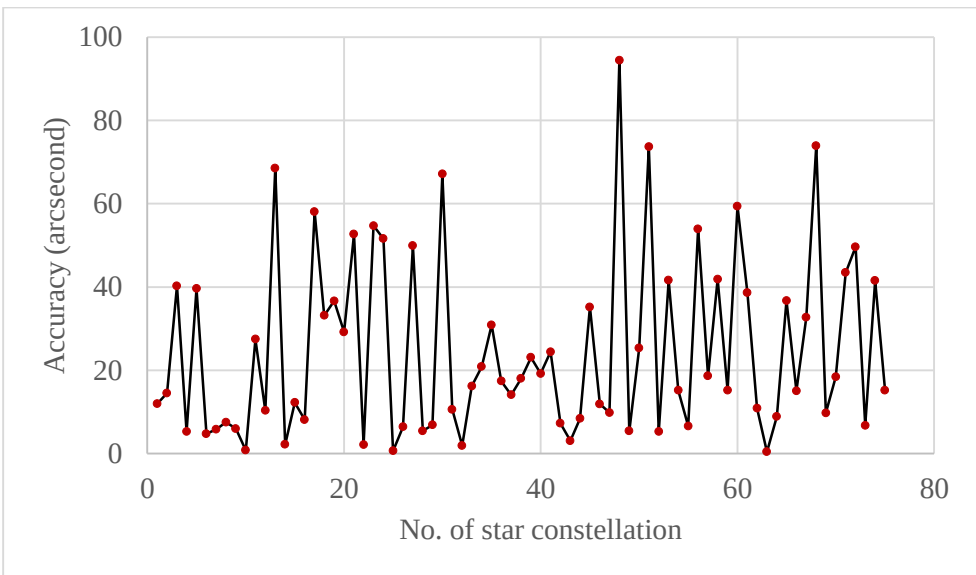

Figure 54: Relative star selection accuracy in yaw axis

Figure 53 the accuracy of the relative star selection is shown in the pitch axis. The accuracy ranges from to 1 to 90 arcsecond and 3σ value in pitch axis 49 arcseconds. Figure 54 the accuracy of the relative star selection is shown in the yaw axis.

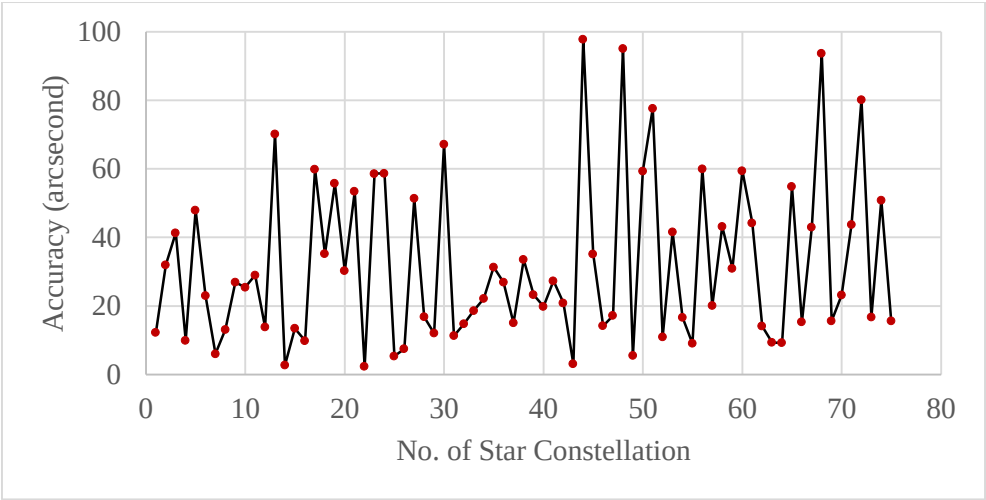

Figure 55: Relative star selection accuracy in boresight axis

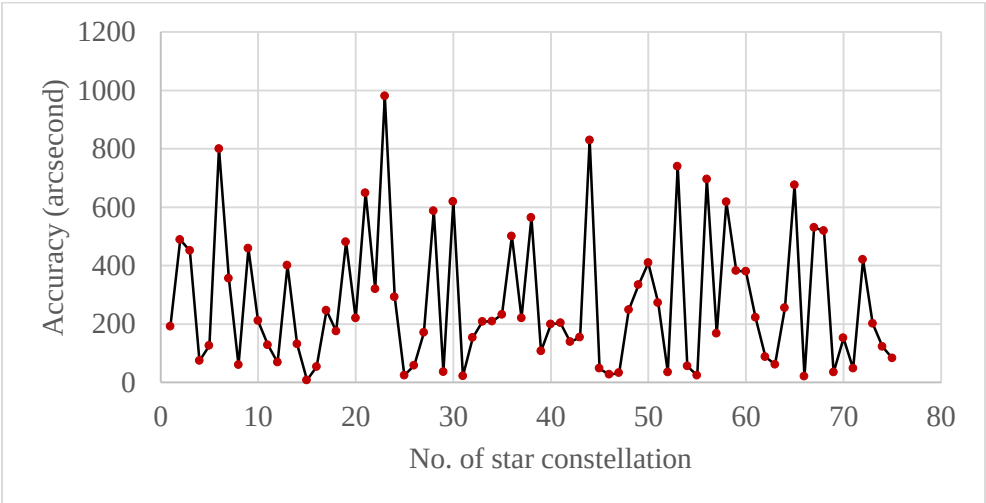

Figure 56: Relative star selection accuracy in roll axis

The accuracy in yaw axis ranges from to 1 to 90 arcsecond. The 3σ value in yaw axis is 64 arcsecond. In Figure 55 the accuracy of the relative star selection is shown in boresight (both pitch and yaw axis). The accuracy ranges from to 1 to 90 arcsecond. The 3σ value in pitch axis 71 arcsecond. In Figure 56 the accuracy of

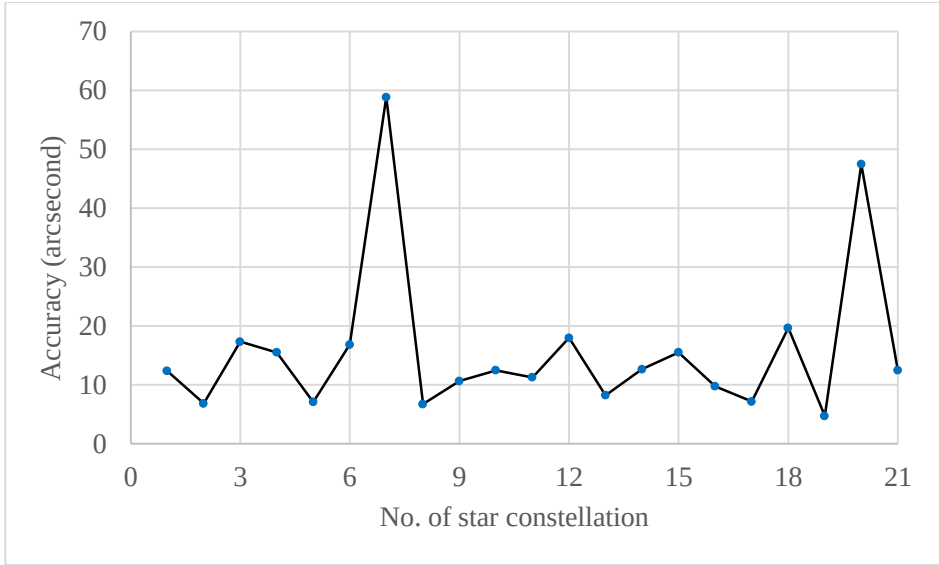

Figure 57: Bright star selection in boresight axis

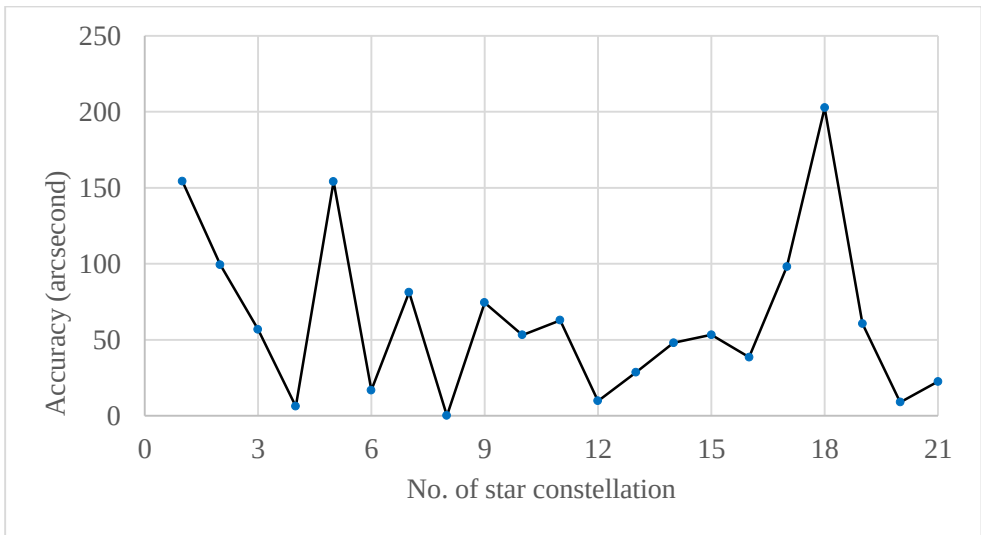

Figure 58: Bright star selection in roll axis

the relative star selection is shown in roll axis. The accuracy ranges from to 1 to 1000 arcsecond. The 3σ value in pitch axis 1071 arcsecond (usually 6 to 15 times lower). In relative star selection all the 75-star constellation are identified successfully. It has 100% success rate. In Figure 57 the accuracy of the bright star

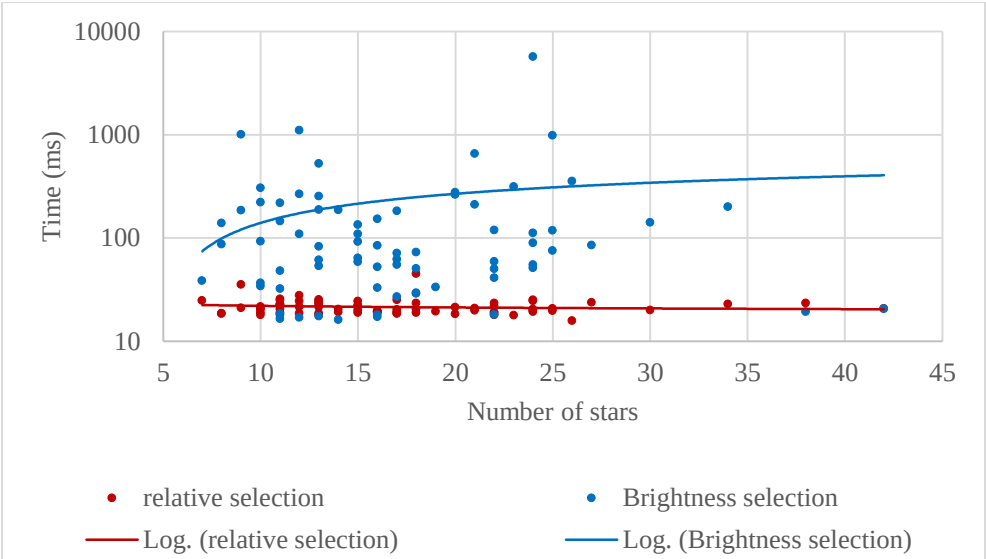

Figure 59: Processing time of relative and bright star selection

selection is shown in boresight (both pitch and yaw axis). The accuracy ranges from to 1 to 60 arcsecond. The 3σ value in pitch axis 61 arcsecond. In Figure 58 the accuracy of the bright star selection is shown in roll axis. The accuracy ranges from to 1 to 200 arcsecond. The 3σ value in roll axis 233 arcsecond. In bright star selection, out of 75-star constellation only 21 are identified successfully and it has 28% success rate. The success rate is reduced because in bright star selection, stars are selected based on brightness. But the distortion error is higher hence stars selected with higher angular distance are misidentified. Hence it has very low success rate.

The graph in Figure 59 shows the processing time of relative and bright star selection. In relative star selection, only four stars are selected irrespective number of stars in the image. Hence the processing time remains constant even if the number of stars in the image is higher. But in bright star selection method the

processing time increases as number of stars in the image increases. The processing time of relative star selection is around 20 ms. But whereas in bright star selection the processing time increases exponentially as a number of stars in the image increases. Table 31 summarizes the results of relative and bright star selection. The relative star selection has high success rate and required low processing time with reasonable accuracy.

Table 31: Performance of relative and bright star selection

Parameters		**Relative Star Selection**	**Bright Star Selection**
Number of star constellation tested		75	75
Number of star constellation identified		75	21
Success rate (%)		100	28
Accuracy (3σ) (arcsecond)	Pitch, Yaw	71	61
	Roll	1071	233
Execution time (ms)	No. of stars: 10	20	140
	No. of stars: 20	21	268
	No. of stars: 30	22	343

The performance of algorithm is tested in laptop i7 processor @ 2.7 GHz. The star simulation and APST algorithm are first implemented in Matlab. In following that the ASPT algorithm are implemented in C++. The image exposure is 100 ms, image processing constitutes 107 ms, relative star selection and identification constitutes 119 ms, TRIAD consumes less than 1 ms. The total time for estimating attitude of satellite is 336 ms. The Matlab and C++ code for star tracker algorithms are given in Appendix A and B. The information about star constellations tested in simulator are given Appendix C.

Chapter 6: Conclusion

This thesis describes about developing the star tracker for nanosatellites. The development constitutes two parts, first part details hardware selection method and second part constitutes development of relative star selection method for APST. The procedure for selecting image sensor and lens from COTS is established. The miniature baffle design and star tracker placement method to eliminate the stray light are determined. The night sky testing validates that selected image sensor and lens detected stars up to 5.8. Due to the radial distortion of the lens (maximum: 0.65%) and centroiding error, angular distance measurement error of the APST varies from 2.08 E-07 to 3.42 E-04. Based on the previous missions and analysis we estimated that angular distance measurement error should be less than 10 E-05 otherwise the stars will be misidentified or unidentified. We identified that the measurement error mainly depends on angular distance between stars. Hence the four stars with lower angular distance is selected which result in successful identification of stars and it named as relative star selection.

The relative star selection has a high success rate (100%) when compared to conventional bright star selection success rate (28 %). The accuracy of the realtive star selection around boresight is 71 arcsecond and 1071 arcsecond in roll. The processing time for relative star selection is constant (22ms) irrespective of number of stars in the image increases. But in the bright star selection method, the processing time increases as exponentially as number of stars in the image increases. Based on the results, the relative star selection is efficient for APST.

The future work includes testing the algorithm in APST hardware in loop. Online testing of APST is required to evaluate its performance. The uneven background illumination test is required to estimate the performance of APST in real scenario. The dynamic thresholding is needed for extracting the star pixels accurately. Gaussian centroiding is required for accurate centroiding of stars. The test setup for distortion estimation and distortion correction algorithm have to be implemented.

References

[1] NASA, www.nasa.gov/content/what-are-smallsats-and-cubesats (cited December 2017)

[2] Space Works, www.spaceworkscommercial.com (cited December 2017)

[3] Planet Labs, www.planet.com/company/approach (cited December 2017)

[4] Sky and Space Global, www.gomspace.com/sky-and-space-global.aspx (cited December 2017)

[5] T. Inamori, N. Sako, T. Hosonuma, J. Takisawa, T.Suehiro, H. Idobata, T. Noumi "Attitude determination and control system for the micro-satellite astrometry missions" 28th ISTS, 2011-d-08, Okinawa, Japan, 2011.

[6] R. Zenick, T.J. McGuire, "Lightweight, low-power coarse star tracker" 17th AIAA/USU Conference on small satellites, SSC03-X-7, Utah, U.S.A, 2003.

[7] B. Bogenberger, "Attitude determination methods" Purdue University, January 2004 (spacecraft design lecture material).

[8] Terma, www.terma.com/space/space-segment/star-trackers, (cited December 2017)

[9] C.Schmidt, CH. Kuhl, K. Michel, "Advance star sensor based on active pixel sensor technology" 6th International ESA GNCS conference, ESA SP-606, Loutraki, Greece, October 2005.

[10] M. Prokhorov, M. Abuberkerov, A. Biryulkov, O.S. Shchikov, M.Tuchin, A. Zakharov.: " Star tracker on a chip", 27th Annual AIAA/USU, SSC13-WK-2, 2013.

[11] T. Segert, S. Engelen, S. Buhl, B. Monna.: "Development of the pico star tracker ST-200 design challenges and the road ahead", 25th Annual AIAA/USU, SSC11-IX-4, 2011.

[12] T. Dzamba, J. Enright, D. Sinclair, K. Amankwah, R. Votel, I. Jovanovic, G. McVittie.: "Success by 1000 improvements: flight qualification of the ST-16 star tracker", 28th Annual AIAA/USU, SSC14-XII-1, 2014.

[13] A.O. Erlank, W.H. Steyn.: "Arcminute attitude estimation for CubeSats with a novel nanostar tracker" IFAC, Vol. 47, 2014, pp.9679-9684

[14] T. Schwaraz.: “Prototyping of a star tracker for pico-satellites, Master thesis, Julius-Maximilians-University, Wuerzburg, Germany, 2015.

[15] Sinclair Interplanetary, www.sinclairinterplanetary.com (cited July 2016)

[16] L. Kazemi, J. Enright, T. Dzamba.: “Improving star tracker centroiding Performance in dynamic imaging conditions” 978-1-4799-5380-6/15, IEE Aerospace conference, Big Sky, Montana, U.S.A, 2015.

[17] M.V. Arbabmir, S.M. Mohammadi, S. Salashour, and F. Somayehee.: “Improving night sky star image processing algorithm for star sensors”, Journal of optical society of America, Vol. 31, No. 4, 2014.

[18] J. Gwanghyeok.: “Autonomous star sensing, pattern identification and attitude determination for spacecraft: An analytical and Experimental study”, Ph.D. Thesis, Texas A&M University, 2001.

[19] B. B. Spratling and D. Mortari.: “A survey on star identification algorithms” Algorithms, 2, 1 (2009), pp. 93-107.

[20] M.D. Shuster and S.D. Oh.: “Three-axis attitude determination from vector observation” Journal of Guidance and Control, Vol.4, No. 1, 1980.

[21] HIPPARCOS-2, www.cosmos.esa.int/web/hipparcos/hipparcos-2 (cited July 2016)

[22] A.I. Zakharov, M.E Prokhorov, M.S. Tuchin.: “Minimum star tracker specification required to achieve a given attitude accuracy” Astrophysical bulletin, 68(2013), pp. 481-493.

[23] M.N. Cannata, M.R. Greene, S.J. Mulligan, V. Popovici.: “Autonomous star-imaging attitude sensor”, York University, Toronto, Canada, 2007.

[24] Mono Camera Sensor Review, Point Grey, www.ptgrey.com (cited March 2016)

[25] M. Marciniak and J.Enright.: “Validating microsatellite star tracker baffle tests”, AIAA 2014-4422, San Diego, California, 2014.

[26] K.M Huffman, R.J. Sedwick, J. Stafford, J. Peverill, and W. Seng.: “Designing star tracker to meet microsatellite requirement”, AIAA 2006-5654, Space ops conference, 2006.

[27] C. Liebe, L. Alkalai, G. Domingo, B. Hancock, D. Hunter, J. Mellstrom, I.

Ruiz, C. Sepulveda, B. Pain.: “Micro APS based star tracker”, IEEE, Vol.5, 2002, pp. 2285-2299.

[28] J. Enright, D. Sinclair, and T. Dzamba.: “The things you can’t ignore: Evolving a sub-arcsecond star tracker”, AIAA/USU, SSC12-X-7, 2012.

[29] T. Dzamba.: “Improving the measurement quality of small satellite star trackers”, Doctoral thesis, University of Ryerson, Canada, 2013.

[30] S. McKechnie.: “General theory of light propagation and imaging through the atmosphere”, Springer, Saint Andrews, Scotland, 2016.

[31] Y. Du, Y.He, H. Chen, W. Xin and B. Xue.: Calculation method of earth-atmosphere stray light illuminance on low-orbit space cameras, Journal of Multimedia, 8, 6 (2013), pp 699-704.

[32] S.F. Zhao, H.T Wang, Y. Wang and C.Y. Ji.: “Space luminous environment adaptability of missile-borne star sensor”, Central South University Press and Springer-Verlag, Berlin, Heidelberg, 2012.

[33] E.C. Fest.: “Stray light analysis and control”, SPIE Press, Bellingham, Washington, USA, 2013.

[34] M.J. Jacobs.: “A low-cost high precision star sensor”, Master Thesis, University of Stellenbosch, South Africa, 1995.

[35] J. Heinisch.: “Light baffle attenuation measurement in the visible”, Journal of Applied Optics, 10, 9 (1971), pp 2016-2020.

[36] V.A. Muruganandan, J.H. Park, A. Maskey, I. Jeung.: “ The estimation of radiation exposure for arcsecond pico star tracker in low earth orbit”, APISAT-2016-C5-5, Toyama, Japan, 2016.

[37] D. Ying, X. Fei, Y. Zheng, “Brightness independent 4-star matching algorithm for lost in space 3-axis attitude acquisition” Tsinghua Science and Technology, Vol. 11, No. 5, Beijing, China, 2006.

[38] D. Mortari “Search less algorithm for star pattern recognition”, The Journal of Astronomical Sciences, 45(2): 179-194, 1997.

[39] A. R. Ayal, Star detection algorithm for estcube-2 star tracker, bachelor thesis, University of Tartu, 2016.

[40] T. Kuwahar, S. Battazzo, Y. Tomioka, Y. Sakamoto, and K. Yoshida, “System

integration of a star sensor for the small earth observation satellite RISING-2" 2011-d-11, 28th ISTS, Okinawa, Japan, 2011.

[41] M. Pal, M.S. Bhat.: "Autonomous star camera calibration and spacecraft attitude determination" Journal of Intelligent Robot System, 79, pp. 323-343, 2015.

[42] D. Mortari, M.A Samaan, C.Bruccoleri, J.L. Junkins.: "The pyramid star identification technique", Journal of the Institute of navigation, Vol.51, No.3, Fall, 2004.

Appendix A: Matlab Code

```
//////image detection////////////
I=rgb2gray(imread('11,c,1.8,100.bmp'));
F1=figure(1);
F1=imshow(I);
impixelregion(F1)
M=sum(sum(I))/(2550*1943);
S=sqrt((sum(sum(I.^2))-2*sum(sum(I*M))+2550*1943*M^2)/(2550*1943));
t=5*S;
[r,c]=find(I>t);
for i=1:length(r)
            I_thresh(r(i),c(i))=I(r(i),c(i));
end
toc
figure(2)
plot(c,r,'.');
figure(3)
[counts,binLocations]=imhist(I);
imhist(I)
F4=figure(4);
F4=imshow(I_thresh);
impixelregion(F4);

////////////grouping/////////////////
cc = bwconncomp(I_thresh,8);
cc_mod.ImageSize=cc.ImageSize;
cc_mod.Connectivity=cc.Connectivity;
j=0;
for i=1:cc.NumObjects(1,1)
      [a,b]=size(cc.PixelIdxList{1,i});
      if a*b>14 && a*b<400
            j=j+1;
          cc_mod.PixelIdxList{1,j}=cc.PixelIdxList{1,i};
      end
end
cc_mod.NumObjects(1,1)=j;
map=[1,1,1];
L = labelmatrix(cc_mod);
RGB_label = label2rgb(L,gray, 'w', 'shuffle');
F6=figure(6);
F6=imshow(RGB_label,'InitialMagnification','fit');
s=regionprops('table',L,I,'weightedcentroid','area','MajorAxisLength','MinorAxisLe
ngth');
```

```
stat=regionprops(L,I,'weightedcentroid','area','MajorAxisLength','MinorAxisLengt
h');
F7=figure(7);
f7=imshow(RGB_label); hold on;
for x = 1: numel(stat)
    plot(stat(x).WeightedCentroid(1),stat(x).WeightedCentroid(2),'ro');
end
stars=table2struct(s);
sc=height(s);

acx=zeros(1,sc);
acy=zeros(1,sc);
for c = 1:sc
cx= stars(c).WeightedCentroid(1)-1296;
cy= stars(c).WeightedCentroid(2)-972;
acx(c)=cx;
acy(c)=cy;
end

fmm= 16.4;
format long ;
f= (2592/5.7)*fmm;
xn= combntns(acx,2);
yn=combntns(acy,2);
sn=size(xn);tx=sn(1);
xn1=xn(:,1); xn2=xn(:,2);
yn1=yn(:,1); yn2=yn(:,2);

ad=zeros(1,tx);
for tn=1:tx
dxy=((xn1(tn)*xn2(tn))+(yn1(tn)*yn2(tn))+f^2)/((sqrt(xn1(tn)^2+yn1(tn)^2+f^2))*
(sqrt(xn2(tn)^2+yn2(tn)^2+f^2)));
ad(tn)=dxy;
end
bp=combntns(1:sc,2);
dr=acosd(ad);
bf=find(dr<0.161);
nbf=numel(bf);

if nbf==1
      bs1=dr(bf(1));
      bsl=find(dr==bs1);
      bsp=bp(bsl,:);
      acx(bsp(1))=[];acy(bsp(2))=[];

end
```

```
if nbf==2

      bs1=dr(bf(1));
      bsl=find(dr==bs1);
      bsp=bp(bsl,:);

      bs2=dr(bf(2));
      bsl2=find(dr==bs2);
      bsp2=bp(bsl2,:);
      bsa=[bsp;bsp2];
      bss=sort(bsa,'descend');
      acx(bss(1,1))=[];acy(bss(1,2))=[];
      acx(bss(2,1))=[];acy(bss(2,2))=[];

end
      if nbf==3

         bs1=dr(bf(1));
      bsl=find(dr==bs1);
      bsp=bp(bsl,:);

        bs2=dr(bf(2));
      bsl2=find(dr==bs2);
      bsp2=bp(bsl2,:);

      bsp3=dr(bf(3));
      bsl3=find(dr==bsp3);
      bsp3=bp(bsl3,:);

      bsa=[bsp;bsp2;bsp3];
      bss=sort(bsa,'descend');
      acx(bss(1,1))=[];acy(bss(1,2))=[];
      acx(bss(2,1))=[];acy(bss(2,2))=[];
      acx(bss(3,1))=[];acy(bss(3,2))=[];

      end
   nostars =numel(acx)
   acx;
   acy;
////////relative star selection////////
fmm= 16.4;
format long ;
f= (2592/5.7)*fmm;
xc= combntns(acx,2);
yc=combntns(acy,2);
```

```
sz=size(xc);rx=sz(1);
xc1=xn(:,1); xc2=xn(:,2);
yc1=yn(:,1); yc2=yn(:,2);adm=zeros(1,rx);
for rn=1:rx
sxy=((xc1(rn)*xc2(rn))+(yc1(rn)*yc2(rn))+f^2)/((sqrt(xc1(rn)^2+yc1(rn)^2+f^2))*
(sqrt(xc2(rn)^2+yc2(rn)^2+f^2)));
adm(rn)=sxy;
end

dc=acosd(adm);
ds=sort(dc);
pL=[find(dc==ds(1)),find(dc==ds(2)),find(dc==ds(3)),find(dc==ds(4)),find(dc==d
s(5)),find(dc==ds(6)),find(dc==ds(7)),find(dc==ds(8)),find(dc==ds(9)),find(dc==d
s(10)),find(dc==ds(11)),find(dc==ds(12)),find(dc==ds(13)),find(dc==ds(14)),find(
dc==ds(15))];
bc=combntns(1:sc,2);
sp=bc(pL,:);
spa=[sp(1,1),sp(1,2),sp(2,1),sp(2,2),sp(3,1),sp(3,2),sp(4,1),sp(4,2),sp(5,1),sp(5,2),s
p(6,1),sp(6,2),sp(7,1),sp(7,2),sp(8,1),sp(8,2),sp(9,1),sp(9,2),sp(10,1),sp(10,2),sp(11
,1),sp(11,2),sp(12,1),sp(12,2),sp(13,1),sp(13,2),sp(14,1),sp(14,2),sp(15,1),sp(15,2)];
spm=mode(spa);spa1=sp(:,1);spa2=sp(:,2);
spf1= find(spa1==spm);
spf2=find(spa2==spm);
ss1=transpose(spa2(spf1));
ss2=transpose(spa1(spf2));
ss=horzcat(ss1,ss2);
s1=spm(1);s2=ss(1);s3=ss(2);s4=ss(3);
AS=[s1,s2,s3,s4];

F7=figure(7);
f7=imshow(RGB_label); hold on;
for x = 1: numel(stat)
    if (x == s1)
        plot(stat(s1).WeightedCentroid(1),stat(s1).WeightedCentroid(2),'rd');
    elseif (x == s2)
        plot(stat(s2).WeightedCentroid(1),stat(s2).WeightedCentroid(2),'rd');
     elseif (x == s3)
         plot(stat(s3).WeightedCentroid(1),stat(s3).WeightedCentroid(2),'rd');
     elseif (x == s4)
        plot(stat(s4).WeightedCentroid(1),stat(s4).WeightedCentroid(2),'rd');
     else
         plot(stat(x).WeightedCentroid(1),stat(x).WeightedCentroid(2),'ro');
     end
end
```

```
//////////4 star matching//////////

  fmm= 16.4;
  format long ;
  f= (2592/5.7)*fmm;

fields.back().push_back(centroid_pixely[i]);
p1=((acx(s1)*acx(s2))+(acy(s1)*acy(s2))+f^2)/((sqrt(acx(s1)^2+acy(s1)^2+f^2))*(
sqrt(acx(s2)^2+acy(s2)^2+f^2)));

p2=((acx(s1)*acx(s3))+(acy(s1)*acy(s3))+f^2)/((sqrt(acx(s1)^2+acy(s1)^2+f^2))*(
sqrt(acx(s3)^2+acy(s3)^2+f^2)));

p3=((acx(s2)*acx(s3))+(acy(s2)*acy(s3))+f^2)/((sqrt(acx(s2)^2+acy(s2)^2+f^2))*(
sqrt(acx(s3)^2+acy(s3)^2+f^2)));

p4=((acx(s1)*acx(s4))+(acy(s1)*acy(s4))+f^2)/((sqrt(acx(s1)^2+acy(s1)^2+f^2))*(
sqrt(acx(s4)^2+acy(s4)^2+f^2)));

p5=((acx(s2)*acx(s4))+(acy(s2)*acy(s4))+f^2)/((sqrt(acx(s2)^2+acy(s2)^2+f^2))*(
sqrt(acx(s4)^2+acy(s4)^2+f^2)));

p6=((acx(s3)*acx(s4))+(acy(s3)*acy(s4))+f^2)/((sqrt(acx(s3)^2+acy(s3)^2+f^2))*(
sqrt(acx(s4)^2+acy(s4)^2+f^2)));

p=[p1, p2, p3,p4,p5,p6];
e= 0.000007; m= 4.96309E-07; q= 0.942007877;
b= floor(((p+e)-q)/m);
t= ceil(((p-e)-q)/m)+1;
np=b-t;
load ('searchm.mat');

kt=[num(t(1),3),num(t(2),3),num(t(3),3),num(t(4),3),num(t(5),3),num(t(6),3)];
kb=[num(b(1),3),num(b(2),3),num(b(3),3),num(b(4),3),num(b(5),3),num(b(6),3)];

a1c=[num(kt(1):kb(1),1),num(kt(1):kb(1),2)];
a2c=[num(kt(2):kb(2),1),num(kt(2):kb(2),2)];
a3c=[num(kt(3):kb(3),1),num(kt(3):kb(3),2)];
a4c=[num(kt(4):kb(4),1),num(kt(4):kb(4),2)];
a5c=[num(kt(5):kb(5),1),num(kt(5):kb(5),2)];
a6c=[num(kt(6):kb(6),1),num(kt(6):kb(6),2)];

id1=(intersect(intersect(a1c,a2c),a4c));
rn1=numel(id1);
if(rn1>0 && rn1<2)
fs1=id1;
```

```
else
  fs1=[];
end

id2=(intersect(intersect(a3c,a5c),a1c));
rn2=numel(id2);
if(rn2>0 && rn2<2)
fs2=id2;
else
  fs2=[];
end

id3=(intersect(intersect(a6c,a3c),a2c));
rn3=numel(id3);
if(rn3>0 && rn3<2)
fs3=id3;
else
  fs3=[];
end

id4=(intersect(intersect(a4c,a5c),a6c));
rn4=numel(id4);
if(rn4>0 && rn4<2)
fs4=id4;
else
  fs4=[];
end

nzv=[fs1,fs2,fs3,fs4];

p1i=num(kt(1):kb(1),1); p1j=num(kt(1):kb(1),2); p2i=num(kt(2):kb(2),1);
p2j=num(kt(2):kb(2),2);
p3i=num(kt(3):kb(3),1); p3j=num(kt(3):kb(3),2); p4i=num(kt(4):kb(4),1);
p4j=num(kt(4):kb(4),2);
p5i=num(kt(5):kb(5),1); p5j=num(kt(5):kb(5),2); p6i=num(kt(6):kb(6),1);
p6j=num(kt(6):kb(6),2);

nfs1=numel(fs1);nfs2=numel(fs2);nfs3=numel(fs3);nfs4=numel(fs4);
if nfs1>0
pc1=find(p1i==fs1(1)); pc2=find(p1j==fs1(1));pc3=find(p2i==fs1(1));
pc4=find(p2j==fs1(1)); pc7=find(p4i==fs1(1)); pc8=find(p4j==fs1(1));
os=[(p1j(pc1)).',(p1i(pc2)).',(p2j(pc3)).',(p2i(pc4)).',(p4j(pc7)).',(p4i(pc8)).'];
sid=[fs1(1),os(1),os(2),os(3)];d
sc1=s1; sc2=s2;sc3=s3; sc4=s4;
elseif nfs2>0
```

```
pc1=find(p1i==fs2(1)); pc2=find(p1j==fs2(1)); pc5=find(p3i==fs2(1));
pc6=find(p3j==fs2(1));pc9=find(p5i==fs2(1)); pc10=find(p5j==fs2(1));
os=[(p1j(pc1)).',(p1i(pc2)).',(p3j(pc5)).',(p3i(pc6)).',(p5j(pc9)).',(p5i(pc10)).'];
sid=[fs2(1),os(1),os(2),os(3)];
sc1=s2; sc2=s1;sc3=s3; sc4=s4;
elseif nfs3>0
pc3=find(p2i==fs3(1)); pc4=find(p2j==fs3(1));pc5=find(p3i==fs3(1));
pc6=find(p3j==fs3(1)); pc11=find(p6i==fs3(1)); pc12=find(p6j==fs3(1));
os=[(p2j(pc3)).',(p2i(pc4)).',(p3j(pc5)).',(p3i(pc6)).',(p6j(pc11)).',(p6i(pc12)).'];
sid=[fs3(1),os(1),os(2),os(3)];
sc1=s3; sc2=s1;sc3=s2; sc4=s4;
elseif nfs4>0
pc7=find(p4i==fs4(1)); pc8=find(p4j==fs4(1));pc9=find(p5i==fs4(1));
pc10=find(p5j==fs4(1)); pc11=find(p6i==fs4(1)); pc12=find(p6j==fs4(1));
os=[(p4j(pc7)).',(p4i(pc8)).',(p5j(pc9)).',(p5i(pc10)).',(p6j(pc11)).',(p6i(pc12)).'];
sid=[fs4(1),os(1),os(2),os(3)];
sc1=s4; sc2=s1;sc3=s2; sc4=s3;
end

hipid = zeros(3,1);
hipid(1) = starcat(sid(1),2);
hipid(2) = starcat(sid(2),2);
hipid(3) = starcat(sid(3),2);
hipid(4) = starcat(sid(4),2);
hipid;

scp=[sc1,sc2,sc3,sc4];
ps1=stars(sc1).Area(1);
ps2=stars(sc2).Area(1);
ps3=stars(sc3).Area(1);
ps4=stars(sc4).Area(1);
aps=[ps1,ps2,ps3,ps4];
mps=max(aps);
fps1=find(aps==mps);
aps(fps1)=0;
mps2=max(aps);
fps2=find(aps==mps2);
bsid=[sid(fps1),sid(fps2)];
bscp=[scp(fps1),scp(fps2)];
```

```
//////////Bright star selection/////////

cc = bwconncomp(I_thresh,8);
cc_mod.ImageSize=cc.ImageSize;
cc_mod.Connectivity=cc.Connectivity;
j=0;
thresh_size=0 ; %this is min number of pixel in star connected
for i=1:cc.NumObjects(1,1)
      [a,b]=size(cc.PixelIdxList{1,i});
      if a*b>14 && a*b<400
            j=j+1;
          cc_mod.PixelIdxList{1,j}=cc.PixelIdxList{1,i};
      end
end

cc_mod.NumObjects(1,1)=j;
map=[1,1,1];
L = labelmatrix(cc_mod);
RGB_label = label2rgb(L,gray, 'w', 'shuffle');
F6=figure(6);
F6=imshow(RGB_label,'InitialMagnification','fit');
sl=regionprops('table',L,I,'weightedcentroid','area');
sla = sortrows(sl,'Area','descend');
stat=regionprops(L,I,'weightedcentroid','area');
F7=figure(7);
f7=imshow(RGB_label); hold on;
for x = 1: numel(stat)
    plot(stat(x).WeightedCentroid(1),stat(x).WeightedCentroid(2),'ro');
end
stars=table2struct(sla);
sc=height(sla);
acx=zeros(1,sc);
acy=zeros(1,sc);
for c = 1:sc
cx= stars(c).WeightedCentroid(1)-1296;
cy= stars(c).WeightedCentroid(2)-972;
acx(c)=cx;
acy(c)=cy;
end

fmm= 16;
format long ;
f= (2592/5.7)*fmm;
accx=combntns(acx,4);accy=combntns(acy,4);
scx=size(accx);sx=scx(1);
ncomb=combntns(1:sc,4);
```

```
cx1=accx(:,1);cx2=accx(:,2);cx3=accx(:,3);cx4=accx(:,4);
cy1=accy(:,1);cy2=accy(:,2);cy3=accy(:,3);cy4=accy(:,4);

cn=1
nis=1;
while nis==1
       bt=0;
     while bt==0
        bst=0;
           while bst==0
  p1=
((cx1(cn)*cx2(cn))+(cy1(cn)*cy2(cn))+f^2)/((sqrt(cx1(cn)^2+cy1(cn)^2+f^2))*(sq
rt(cx2(cn)^2+cy2(cn)^2+f^2)));
  p2=
((cx1(cn)*cx3(cn))+(cy1(cn)*cy3(cn))+f^2)/((sqrt(cx1(cn)^2+cy1(cn)^2+f^2))*(sq
rt(cx3(cn)^2+cy3(cn)^2+f^2)));
  p3=
((cx2(cn)*cx3(cn))+(cy2(cn)*cy3(cn))+f^2)/((sqrt(cx2(cn)^2+cy2(cn)^2+f^2))*(sq
rt(cx3(cn)^2+cy3(cn)^2+f^2)));
  p4=
((cx2(cn)*cx3(cn))+(cy2(cn)*cy3(cn))+f^2)/((sqrt(cx1(cn)^2+cy1(cn)^2+f^2))*(sq
rt(cx3(cn)^2+cy3(cn)^2+f^2)));
  p5=
((cx1(cn)*cx4(cn))+(cy1(cn)*cy4(cn))+f^2)/((sqrt(cx1(cn)^2+cy1(cn)^2+f^2))*(sq
rt(cx4(cn)^2+cy4(cn)^2+f^2)));
  p6=
((cx3(cn)*cx4(cn))+(cy3(cn)*cy4(cn))+f^2)/((sqrt(cx3(cn)^2+cy3(cn)^2+f^2))*(sq
rt(cx4(cn)^2+cy4(cn)^2+f^2)));

load ('searchm.mat');
cs=size(num);csr=cs(1);
p=[p1, p2, p3,p4,p5,p6];
e= 0.000007; m= 4.96309E-07; q= 0.942007877;% (e =0.0000064 to 0.0000075)
b= floor(((p+e)-q)/m);
t= ceil(((p-e)-q)/m)+1;

bs=any(b<0);ts=any(t<0);
bsc=[bs,ts]; bst=all(bsc==0);
if bst==1
      cn;
else
cn=cn+1;
end
           end
bc=(b(1,:)<csr);tc=(t(1,:)<csr);
bz=all(bc);tz=all(tc);
```

```
bzc=[bz,tz];bt=all(bzc);
if bt==1
       cn;
else
cn=cn+1;
end
     end

np=b-t;

kt=[num(t(1),3),num(t(2),3),num(t(3),3),num(t(4),3),num(t(5),3),num(t(6),3)];
kb=[num(b(1),3),num(b(2),3),num(b(3),3),num(b(4),3),num(b(5),3),num(b(6),3)];

a1c=[num(kt(1):kb(1),1),num(kt(1):kb(1),2)];
a2c=[num(kt(2):kb(2),1),num(kt(2):kb(2),2)];
a3c=[num(kt(3):kb(3),1),num(kt(3):kb(3),2)];
a4c=[num(kt(4):kb(4),1),num(kt(4):kb(4),2)];
a5c=[num(kt(5):kb(5),1),num(kt(5):kb(5),2)];
a6c=[num(kt(6):kb(6),1),num(kt(6):kb(6),2)];

id1=(intersect(intersect(a1c,a2c),a4c));
rn1=numel(id1);
if(rn1>0 && rn1<2)
fs1=id1;
else
  fs1=[];
end

id2=(intersect(intersect(a3c,a5c),a1c));
rn2=numel(id2);
if(rn2>0 && rn2<2)
fs2=id2;
else
  fs2=[];
end

id3=(intersect(intersect(a6c,a3c),a2c));
rn3=numel(id3);
if(rn3>0 && rn3<2)
fs3=id3;
else
  fs3=[];
end

id4=(intersect(intersect(a4c,a5c),a6c));
rn4=numel(id4);
```

```
if(rn4>0 && rn4<2)
fs4=id4;
else
  fs4=[];
end

nzv=[fs1,fs2,fs3,fs4];
nis=isempty(nzv);
if cn>sx
      break
end
cn=cn+1

end

p1i=num(kt(1):kb(1),1); p1j=num(kt(1):kb(1),2); p2i=num(kt(2):kb(2),1);
p2j=num(kt(2):kb(2),2);
p3i=num(kt(3):kb(3),1); p3j=num(kt(3):kb(3),2); p4i=num(kt(4):kb(4),1);
p4j=num(kt(4):kb(4),2);
p5i=num(kt(5):kb(5),1); p5j=num(kt(5):kb(5),2); p6i=num(kt(6):kb(6),1);
p6j=num(kt(6):kb(6),2);

s1=ncomb(cn,1); s2=ncomb(cn,2); s3=ncomb(cn,3); s4=ncomb(cn,4);
nfs1=numel(fs1);nfs2=numel(fs2);nfs3=numel(fs3);nfs4=numel(fs4);
if nfs1>0
      pc1=find(p1i==fs1(1)); pc2=find(p1j==fs1(1));pc3=find(p2i==fs1(1));
pc4=find(p2j==fs1(1)); pc7=find(p4i==fs1(1)); pc8=find(p4j==fs1(1));
      os=[(p1j(pc1)).',(p1i(pc2)).',(p2j(pc3)).',(p2i(pc4)).',(p4j(pc7)).',(p4i(pc8)).'];
      sid=[fs1(1),os(1),os(2),os(3)];
      sc1=s1; sc2=s2;sc3=s3; sc4=s4;
elseif nfs2>0
      pc1=find(p1i==fs2(1)); pc2=find(p1j==fs2(1)); pc5=find(p3i==fs2(1));
pc6=find(p3j==fs2(1));pc9=find(p5i==fs2(1)); pc10=find(p5j==fs2(1));
      os=[(p1j(pc1)).',(p1i(pc2)).',(p3j(pc5)).',(p3i(pc6)).',(p5j(pc9)).',(p5i(pc10)).'];
      sid=[fs2(1),os(1),os(2),os(3)];
      sc1=s2; sc2=s1;sc3=s3; sc4=s4;
elseif nfs3>0
      pc3=find(p2i==fs3(1)); pc4=find(p2j==fs3(1));pc5=find(p3i==fs3(1));
pc6=find(p3j==fs3(1)); pc11=find(p6i==fs3(1)); pc12=find(p6j==fs3(1));
     os=[(p2j(pc3)).',(p2i(pc4)).',(p3j(pc5)).',(p3i(pc6)).',(p6j(pc11)).',(p6i(pc12)).'];
      sid=[fs3(1),os(1),os(2),os(3)];
      sc1=s3; sc2=s1;sc3=s2; sc4=s4;
elseif nfs4>0
      pc7=find(p4i==fs4(1)); pc8=find(p4j==fs4(1));pc9=find(p5i==fs4(1));
pc10=find(p5j==fs4(1)); pc11=find(p6i==fs4(1)); pc12=find(p6j==fs4(1));
```

```
os=[(p4j(pc7)).',(p4i(pc8)).',(p5j(pc9)).',(p5i(pc10)).',(p6j(pc11)).',(p6i(pc12)).'];
      sid=[fs4(1),os(1),os(2),os(3)];
      sc1=s4; sc2=s1;sc3=s2; sc4=s3;
end

%hipid = zeros(3,1);
%hipid(1) = starcat(sid(1),2);
%hipid(2) = starcat(sid(2),2);
%hipid(3) = starcat(sid(3),2);
%hipid(4) = starcat(sid(4),2);
%hipid;

scp=[sc1,sc2,sc3,sc4];
ps1=stars(sc1).Area(1);
ps2=stars(sc2).Area(1);
ps3=stars(sc3).Area(1);
ps4=stars(sc4).Area(1);
aps=[ps1,ps2,ps3,ps4];
mps=max(aps);
fps1=find(aps==mps);
aps(fps1)=0;
mps2=max(aps);
fps2=find(aps==mps2);
bsid=[sid(fps1),sid(fps2)];
bscp=[scp(fps1),scp(fps2)];

////////////////TRIAD////////////////////////

load('mcfm.mat');
v1=num2(bsid(1),:);     %true s1
v2=num2(bsid(2),:);     %true s2

w1temp = [-acx(bscp(1)), -acy(bscp(1)), f];
w2temp = [-acx(bscp(2)), -acy(bscp(2)), f];
w1 = w1temp/norm(w1temp);
w2 = w2temp/norm(w2temp);

if1=v1;
if2=(cross(v1,v2))./(norm(cross(v1,v2)));
if3=(cross(v1,cross(v1,v2)))./(norm(cross(v1,v2)));

sf1=w1/norm(w1);
sf2=(cross(w1,w2))./(norm(cross(w1,w2)));
sf3=(cross(w1,cross(w1,w2)))./(norm(cross(w1,w2)));
```

```
sf = [sf1' sf2' sf3'];
ifr = [if1' if2' if3'];
rm=sf*ifr';
quat=rotm2quat(rm);
eul = rotm2eul(rm, 'ZYZ')*180/pi;

err = [-eul(3) - true_ra, -eul(2) - (90-true_de), -eul(1) - true_rot];

if (err(1) >= 180)
      err(1) = err(1) - 360;
elseif (err(1) <= -180)
      err(1) = err(1) + 360;
end

if (err(2) >= 180)
      err(2) = err(2) - 360;
elseif (err(2) <= -180)
      err(2) = err(2) + 360;
end

if (err(3) >= 180)
      err(3) = err(3) - 360;
elseif (err(3) <= -180)
      err(3) = err(3) + 360;
end

errc=[err(1),err(2),err(3)]

err_arc = errc*3600


//////star simulator/////

function y = starsim5(RA, DE, ROT, fov1, fov2, a, b, apix, bpix, f, mag, nois)

% RA DE is center of cmos

% a b is size of cmos a * b
%                 ----------
%             |                    |
%        b    |                    |
%             |                    |
%                 ----------
%                         a

% pix : number of pixels
```

```
% v2 uses centroid + size to determine which pixels to be bright

    global starcat;

    monostar = uint8(zeros(bpix, apix))+ nois * uint8(2.5*randn(bpix,apix)+4);

    count = 0;
    %RA given 0~360
    %DE given -90 ~ 90
    for i = 1:1:size(starcat,1)

        %anomaly when e.g. RA = 359.9, starRA = 0.1;
        %RA - starRA = 359.8, actually -0.2
        delRA = RA-starcat(i,4);
        if (delRA >= 180)
            delRA = delRA - 360;
        elseif (delRA <= -180)
            delRA = delRA + 360;
        end

        %coord transform Z to reference. Look Z and take picture

        Z = [cos(RA*pi/180) sin(RA*pi/180) 0; ...
            -sin(RA*pi/180) cos(RA*pi/180) 0; ...
            0 0 1];

        Y = [cos((90-DE)*pi/180) 0 -sin((90-DE)*pi/180); ...
            0 1 0; ...
            sin((90-DE)*pi/180) 0 cos((90-DE)*pi/180)];


        %X = [1 0 0; ...
        %       0 cos(180*pi/180) sin(180*pi/180); ...
        %       0 -sin(180*pi/180) cos(180*pi/180)];
        Z2 = [cos(ROT*pi/180) sin(ROT*pi/180) 0; ...
            -sin(ROT*pi/180) cos(ROT*pi/180) 0; ...
            0 0 1];

        newpos = Z2 * Y * Z * starcat(i,8:10)';

        (Z2 * Y * Z)'

        %anomaly when e.g. DE = 89.9, starDE = -89.9;
        %RA - starRA = 179.8, actually -0.2
        delDE = DE-starcat(i,5);
        if (delDE >= 90)
```

```
            delDE = delDE - 180;
        elseif (delDE <= -90)
            delDE = delDE + 180;
        end

        if abs(delRA) < fov1
            if abs(delDE) < fov2
                count = count +1;

                x = -f/newpos(3) * newpos(1);
                y = -f/newpos(3) * newpos(2);

                xpix = x/a * apix;
                ypix = y/b * bpix;

                %add lens distortion model
                radpix = sqrt(xpix^2+ypix^2);
                x = -f/newpos(3) * (newpos(1) + nois * 0.5e10*radpix^1.8179);
                y = -f/newpos(3) * (newpos(2) + nois * 0.5e10*radpix^1.8179);
                xpix = x/a * apix;
                ypix = y/b * bpix;
                %xpix = xpix + 0.5e-10*radpix^1.8179;
                %ypix = ypix + 0.5e-10*radpix^1.8179;

                %pixel number modeling
                npix = -1.2577 * starcat(i,3)^4 ...
                    +17.042 * starcat(i,3)^3    ...
                    -67.096 * starcat(i,3)^2 ...
                    +8.474 * starcat(i,3) ...
                    +312.72;
                rpix = sqrt(npix/pi);

  for ipix = -rpix:1:rpix
     for jpix = -rpix:1:rpix
         if (sqrt((ipix)^2+(jpix)^2) <= rpix)
            %pixel mapping
             ximpix = round(apix/2 + xpix +ipix);
             yimpix = round(bpix/2 + ypix +jpix);
                if ximpix <= 0 || ximpix >= apix || yimpix <= 0 || yimpix >= bpix
                    break;
                    end
  monostar(yimpix, ximpix) = uint8((0.8*2.5*255*(mag - starcat(i,3))
*normpdf(sqrt((ipix)^2+(jpix)^2),0,rpix/2.5)));
                                end
                            end
                        end
```

```
                end
            end
        end

        %at this point, +Y is down, +X is right
        %count
        %monostar = imrotate(monostar,180);
       %F = imshow(monostar);
        %impixelregion(F);

        imwrite(monostar,'sim.png');

        y=0;
end

%     star catalog simulator
%     made by Ji Hyun Park
%     snug88@snu.ac.kr
%     distribute as required
clear;
clc;
close all;

w = warning ('off','all');
exist starcat;

if ans == 0
        global starcat
        starcat = xlsread('HIP2-17.xlsx') ;
end
%fov1 = 19.6;
fov1 = 19.6;
%fov2 = 19.6;
fov2 = 15;
%a = 4.28;
%a = 4.28;
a=5.7;
b = 1944/2592*5.7;%4.28;
%b = 5.7;
%apix = 1944;
apix = 2592;
bpix = 1944;
%bpix = 2592;
mag = 6.5; %larger mag, darker
%under mag, cannot be seen
```

```
true_ra =14.17;
true_de =60.71;
true_rot=180;

starsim5(true_ra, true_de, true_rot, fov1, fov2, a, b, apix, bpix, 16, mag,1);
```

Appendix B: C++ Code

```
#include <iostream>
#include <vector>
#include <map>
#include <chrono>
#include "csv.h"
#include "csv.cpp"
#include "catalog.h"
using namespace std;
using namespace chrono;
const float PI_f = 4*atanf(1.0f);
const double PI_d = 4*atan(1.0);

int32_t row = 0;
int32_t column = 0;
vector<float> centroid_x;
vector<float> centroid_y;

void findStar(const char* filename, int32_t sigma, int32_t clusterMinSize, int32_t
clusterMaxSize, bool saveCSV, bool debugMode);
void getAttitude(int32_t pairMaxNumber, float minimumSeparation, double fmm,
double err, double slope, double yintercept, bool saveCSV, bool debugMode);

int main(void)
{
   findStar("sim24bit.bmp", 5, 14, 400, false, true);
  //findStar("11,vft,1.8,100.bmp", 5, 14, 250, false, true);
  //findStar("11 c 1.8 100.bmp", 5, 14, 250, false, true);
  //findStar("test.bmp", 1, 5, 40000, true, true);

   getAttitude(15, 0.16f, 16.0, 9e-6, 4.96309e-7, 0.942007877, true, true);
  // for c~ use 16.40
  // for v~ use 16.27
  // err 4~9e-6 (7 is average)
  // sim (fmm 16, err 7e-6, clusterSize 14~400)
   return 0;
}

void getAttitude(int32_t pairMaxNumber, float minimumSeparation, double fmm,
double err, double slope, double yintercept, bool saveCSV, bool debugMode)
{
  //5.selection
   cout << "5. selection" << endl;
   steady_clock::time_point begin = steady_clock::now();
```

```
vector<int32_t> pairEuclidean_;
vector<int32_t> pairIndex1_;
vector<int32_t> pairIndex2_;
vector<int32_t> pairEuclidean;
vector<int32_t> pairIndex1;
vector<int32_t> pairIndex2;
int32_t numberofCluster = centroid_x.size();
int32_t index = 0;
for (int32_t i = 0; i < numberofCluster - 1; i++)
{
  for (int32_t j = i + 1; j < numberofCluster; j++)
  {
    float deltaX = centroid_x[i] - centroid_x[j];
    float deltaY = centroid_y[i] - centroid_y[j];
    pairEuclidean_.push_back((int32_t)(deltaX*deltaX + deltaY*deltaY));
    pairIndex1_.push_back(i);
    pairIndex2_.push_back(j);
    }
  }
//sort in ascending order (bubble sort)
  int32_t numberofPair = pairEuclidean_.size();
  for (int32_t i = 0; i < pairMaxNumber && i < numberofPair; i++)
  {
    for (int32_t j = numberofPair - 1; j > i; j--)
    {
      if (pairEuclidean_[j] < pairEuclidean_[j - 1])
      {
        int32_t swapEuclidean = pairEuclidean_[j - 1];
        int32_t swapIndex1 = pairIndex1_[j - 1];
        int32_t swapIndex2 = pairIndex2_[j - 1];
        pairEuclidean_[j - 1] = pairEuclidean_[j];
        pairIndex1_[j - 1] = pairIndex1_[j];
        pairIndex2_[j - 1] = pairIndex2_[j];
        pairEuclidean_[j] = swapEuclidean;
        pairIndex1_[j] = swapIndex1;
        pairIndex2_[j] = swapIndex2;
      }
    }
    float x1 = centroid_x[pairIndex1_[i]];
    float y1 = centroid_y[pairIndex1_[i]];
    float x2 = centroid_x[pairIndex2_[i]];
    float y2 = centroid_y[pairIndex2_[i]];
    float f = column/5.7*fmm;
    float angularSeparation = 180.0f/PI_f*acosf((x1*x2 + y1*y2 + f*f)/
                                  (sqrt(x1*x1 + y1*y1 + f*f)* sqrt(x2*x2 + y2*y2 +
                                  f*f))); if (angularSeparation > minimumSeparation)
```

```
    {
      pairIndex1.push_back(pairIndex1_[i]);
      pairIndex2.push_back(pairIndex2_[i]);
      pairEuclidean.push_back((int32_t)sqrt(pairEuclidean_[i]));
    }
    else pairMaxNumber++;
  }
//find all possible pyramids
  vector<int32_t> frequency(numberofCluster, 0);
  vector<vector<int32_t>> pyramid;
  pairMaxNumber = pairIndex1.size();
  for (int32_t i = 0; i < pairMaxNumber && i < numberofPair; i++)
  {
    frequency[pairIndex1[i]]++;
    frequency[pairIndex2[i]]++;
  }
  for (int32_t i = 0; i < numberofCluster; i++)
  {
    if (frequency[i] > 2) // 3 pairs (or more) with a commmon element i forms a
    pyramid (or more)
    {
      vector<int32_t> pyramidElement;
      for (int32_t j = 0; j < pairMaxNumber && j < numberofPair; j++)
      {
        if (i == pairIndex1[j])pyramidElement.push_back (pairIndex2[j]);
        else if (i == pairIndex2[j]) pyramidElement.push_back(pairIndex1[j]);
      }
      for (int32_t j = 0; j < frequency[i] - 2; j++)
      {
        for (int32_t k = j + 1; k < frequency[i] - 1; k++)
        {
          for (int32_t l = k + 1; l < frequency[i]; l++)
          {
            vector<int32_t> entry;
            entry.push_back(i);
            entry.push_back(pyramidElement[j]);
            entry.push_back(pyramidElement[k]);
            entry.push_back(pyramidElement[l]);
            for (int32_t l1 = 0; l1 < 4; l1++) // entry element in ascending order
            {
              for (int32_t l2 = 3; l2 > l1; l2--)
              {
                if (entry[l2] < entry[l2 - 1])
                {
                  int32_t swap = entry[l2 - 1];
```

```
                entry[l2 - 1] = entry[l2];
                entry[l2] = swap;
              }
             }
            }
            pyramid.push_back(entry);
           }
          }
         }
        }
       }
      //post processing
       vector<vector<int32_t>> pyramid_sorted;
       for (int32_t i = 0; i < numberofCluster - 3; i++)
       {
        for (int32_t j = i + 1; j < numberofCluster - 2; j++)
        {
         for (int32_t k = j + 1; k < numberofCluster - 1; k++)
         {
          for (int32_t l = k + 1; l < numberofCluster; l++)
          {
           int32_t m = 0;
           bool foundMatch = false;
           while (m < pyramid.size() && !foundMatch)
           {
            if (pyramid[m][0] == i && pyramid[m][1] == j && pyramid[m][2] == k
            && pyramid[m][3] == l)
            {
             pyramid_sorted.push_back(vector<int32_t>());
             pyramid_sorted.back().push_back(i);
             pyramid_sorted.back().push_back(j);
             pyramid_sorted.back().push_back(k);
             pyramid_sorted.back().push_back(l);
             foundMatch = true;
             m++;
            }
           }
          }
         }
        }
       //find the best pyramid (smallest pyramid)
        int32_t pyramidEuclidMin = row*column;
        int32_t pyramidEuclidMinIndex = -1;
        for (int32_t i = 0; i < pyramid_sorted.size(); i++)
        {
         int32_t pyramidEuclidSum = 0;
```

```
  for (int32_t j = 0; j < 3; j++)
  {
    for (int32_t k = j + 1; k < 4; k++)
    {
      int32_t myID1 = pyramid_sorted[i][j];
      int32_t myID2 = pyramid_sorted[i][k];
      float deltaX = centroid_x[myID1] - centroid_x[myID2];
      float deltaY = centroid_y[myID1] centroid_y[myID2];
      pyramidEuclidSum += (int32_t)sqrtf(deltaX*deltaX + deltaY*deltaY);
    }
  }
  if (pyramidEuclidSum < pyramidEuclidMin)
  {
    pyramidEuclidMin = pyramidEuclidSum;
    pyramidEuclidMinIndex = i;
  }
  pyramid_sorted[i].push_back(pyramidEuclidSum);
}
steady_clock::time_point end = steady_clock::now();
cout << "elapsed time: " << duration_cast<milliseconds>(end - begin).count()
<< "ms" << endl;
if (saveCSV)
{
  cout << "saving results in csv format..." << endl;
  csv<int32_t> csvout("5pairs_ascending.csv");
  vector<string> header;
  vector<vector<int32_t>> fields;
  for (int32_t i = 0; i < 2; i++) header.push_back("myID " + to_string(i + 1));
  header.push_back("Euclidean distance");
  for (int32_t i = 0; i < pairIndex1.size(); i++)
  {
    fields.push_back(vector<int32_t>());
    fields.back().push_back(pairIndex1[i]);
    fields.back().push_back(pairIndex2[i]);
    fields.back().push_back(pairEuclidean[i]);
  }
  csvout.write(header, fields);
  cout << "saved 5pyramid_candidates_myID.csv" << endl;
}
if (saveCSV)
{
  cout << "saving results in csv format..." << endl;
  csv<int32_t> csvout("5pyramid_candidates_myID.csv");
  vector<string> header;
  for (int32_t i = 0; i < 4; i++) header.push_back("myID " + to_string(i + 1));
  header.push_back("pyramid Euclid sum");
```

```
      csvout.write(header, pyramid_sorted);
      cout << "saved 5pyramid_candidates_myID.csv" << endl;
    }
// 6. catalog matching (k-vector search)
    cout << "6. catalog matching" << endl;
    begin = steady_clock::now();
    double f = column/5.7*fmm; // focal length in pixels
    double pairSpherical[6];
    int32_t lowerIndex[6];
    int32_t upperIndex[6];
    int32_t count = 0;
    vector<int32_t> pyramid_myID = pyramid_sorted[pyramidEuclidMinIndex];
  //find index range
    for (int32_t i = 0; i < 3; i++)
    {
      for (int32_t j = i + 1; j < 4; j++)
      {
        double x1 = centroid_x[pyramid_myID[i]];
        double y1 = centroid_y[pyramid_myID[i]];
        double x2 = centroid_x[pyramid_myID[j]];
        double y2 = centroid_y[pyramid_myID[j]];
        pairSpherical[count] = (x1*x2 + y1*y2 + f*f)/(sqrt(x1*x1 + y1*y1 + f*f) *
                                  sqrt (x2*x2 + y2*y2 + f*f));
        upperIndex[count] = kvector[(int32_t)((pairSpherical[count] +err –
                                yintercept)/ slope) - 1][2]; // upper bound (excluded)
        lowerIndex[count] = kvector[(int32_t)((pairSpherical[count] - err –
                                yintercept) / slope) + 1][2]; // lower bound (included)
        count++;
      }
    }
// find pyramid candidates
    bool pairID[2647][6] = { false, };
    int32_t pairIDcount[2647] = { 0, };
    for (int32_t i = 0; i < 6; i++)
    {
      for (int32_t j = lowerIndex[i]; j < upperIndex[i]; j++)
      {
        if (!pairID[kvector[j][0]][i])
        {
          pairID[kvector[j][0]][i] = true;
          pairIDcount[kvector[j][0]]++;
        }
        if (!pairID[kvector[j][1]][i])
        {
          pairID[kvector[j][1]][i] = true;
          pairIDcount[kvector[j][1]]++;
```

```
    }
  }
}
int32_t pairPattern[4][3] =
{
  0, 1, 2,
  0, 3, 4,
  1, 3, 5,
  2, 4, 5
};
vector<vector<int32_t>> pyramid_starID;
for (int32_t i = 0; i < 2647; i++)
{
  int32_t cond = -1;
  if (pairID[i][0] && pairID[i][1] && pairID[i][2] && !pairID[i][3]
    &&    !pairID[i][4] && !pairID[i][5]) cond = 0;
  else if (pairID[i][0] && !pairID[i][1] && !pairID[i][2] && pairID[i][3] &&
    pairID[i][4] && !pairID[i][5]) cond = 1;
  else if (!pairID[i][0] && pairID[i][1] && !pairID[i][2] && pairID[i][3]
    && !pairID[i][4] && pairID[i][5]) cond = 2;
  else if (!pairID[i][0] && !pairID[i][1] && pairID[i][2] && !pairID[i][3] &&
    pairID[i][4] && pairID[i][5]) cond = 3;
  if (!(cond < 0))
  {
    vector<int32_t> entry(4);
    entry[cond] = i; // common ID
    bool existDuplicate = false;
    for (int32_t j = 0; j < 3; j++)
    {
      int32_t count = 0;
      for (int32_t k = lowerIndex[pairPattern[cond][j]]; k <
                          upperIndex[pairPattern[cond][j]]; k++)
      {
        if (kvector[k][0] == i)
        {
          if (cond > j) entry[j] = kvector[k][1];
          else entry[j + 1] = kvector[k][1];
          count++;
        }
        else if (kvector[k][1] == i)
        {
          if (cond > j) entry[j] = kvector[k][0];
          else entry[j + 1] = kvector[k][0];
          count++;
        }
      }
```

```
            if (count > 1) existDuplicate = true;
          }
          if (!existDuplicate) pyramid_starID.push_back(entry);
        }
      }
      end = steady_clock::now();
      cout << "elapsed time: " << duration_cast<milliseconds>(end –
        begin).count() << "ms" << endl;
      if (saveCSV)
      {
        cout << "saving results in csv format..." << endl;
        csv<int32_t> csvout("6pyramid_starID.csv");
        vector<string> header;
       for (int32_t i = 0; i < 4; i++) header.push_back("starID" + to_string(i + 1));
          csvout.write(header, pyramid_starID);
        cout << "saved 6pyramid_starID.csv" << endl;
      }
      if (saveCSV)
      {
        cout << "saving results in csv format..." << endl;
        csv<string> csvout("6pair_candidates_starID.csv");
        vector<string> header;
        vector<vector<string>> fields;
        header.push_back("pyramid pair"); header.push_back("spherical
          distance");
        header.push_back("starID 1"); header.push_back("starID 2");
        for (int32_t i = 0; i < 6; i++)
        {
          for (int32_t j = lowerIndex[i]; j < upperIndex[i]; j++)
          {
            fields.push_back(vector<string>());
            fields.back().push_back(to_string(i + 1));
            fields.back().push_back(to_string(pairSpherical[i]));
            fields.back().push_back(to_string(kvector[j][0]));
            fields.back().push_back(to_string(kvector[j][1]));
          }
        }
       csvout.write(header, fields);
        cout << "saved 6pair_candidates_starID.csv" << endl;
      }
// 7. TRIAD
      cout << "7. TRIAD" << endl;
      begin = steady_clock::now();
// find skyCoord & pixelCoord
      double skyCoord1[3];
      double skyCoord2[3];
```

```
    for (int32_t i = 0; i < 3; i++)
    {
      skyCoord1[i] = mcf[pyramid_starID[0][0] - 1][i];
      skyCoord2[i] = mcf[pyramid_starID[0][1] - 1][i];
    }
/*double pixelCoord1[3];
    double pixelCoord2[3];
    double sfx1 = atan(centroid_x[pyramid_myID[0]]/f);
    double sfy1 = atan(centroid_y[pyramid_myID[0]]/f*cos(sfx1));
    double sfx2 = atan(centroid_x[pyramid_myID[1]]/f);
    double sfy2 = atan(centroid_y[pyramid_myID[1]]/f*cos(sfx2));
    pixelCoord1[0] = -sin(sfx1)*cos(sfy1); pixelCoord1[1] =
                        cos(sfx1)*cos(sfy1); pixelCoord1[2] = -sin(sfy1);
    pixelCoord2[0] = -sin(sfx2)*cos(sfy2); pixelCoord2[1] =
                        cos(sfx2)*cos(sfy2); pixelCoord2[2] = -sin(sfy2);*/
    double pixelCoord1[3] = {-centroid_x[pyramid_myID[0]], -
                                centroid_y[pyramid_myID[0]], f};
    double pixelCoord2[3] = {-centroid_x[pyramid_myID[1]], -
                                centroid_y[pyramid_myID[1]], f};
  //normalize
    {
      double norm = 0.0;
      for (int32_t i = 0; i < 3; i++) norm += pixelCoord1[i]*pixelCoord1[i];
      norm = sqrt(norm);
      for (int32_t i = 0; i < 3; i++) pixelCoord1[i] /= norm;
    }
    {
      double norm = 0.0;
      for (int32_t i = 0; i < 3; i++) norm += pixelCoord2[i]*pixelCoord2[i];
      norm = sqrt(norm);
      for (int32_t i = 0; i < 3; i++) pixelCoord2[i] /= norm;
    }
  //find ifn & sfn
    double ifn[3][3];
    double sfn[3][3];
    for (int32_t i = 0; i < 3; i++) ifn[i][0] = skyCoord1[i];
    ifn[0][1] = skyCoord1[1]*skyCoord2[2] - skyCoord2[1]*skyCoord1[2];
    ifn[1][1] = skyCoord2[0]*skyCoord1[2] - skyCoord1[0]*skyCoord2[2];
    ifn[2][1] = skyCoord1[0]*skyCoord2[1] - skyCoord2[0]*skyCoord1[1];
    ifn[0][2] = ifn[1][0]*ifn[2][1] - ifn[1][1]*ifn[2][0];
    ifn[1][2] = ifn[0][1]*ifn[2][0] - ifn[0][0]*ifn[2][1];
    ifn[2][2] = ifn[0][0]*ifn[1][1] - ifn[0][1]*ifn[1][0];
    for (int32_t i = 1; i < 3; i++) // normalize
    {
      double norm = 0;
      for (int32_t j = 0; j < 3; j++) norm += ifn[j][i]*ifn[j][i];
```

```
  norm = sqrt(norm);
  for (int32_t j = 0; j < 3; j++) ifn[j][i] /= norm;
}
for (int32_t i = 0; i < 3; i++) sfn[i][0] = pixelCoord1[i];
sfn[0][1] = pixelCoord1[1]*pixelCoord2[2] - pixelCoord2[1] *
            pixelCoord1[2];
sfn[1][1] = pixelCoord2[0]*pixelCoord1[2] - pixelCoord1[0] *
            pixelCoord2[2];
sfn[2][1] = pixelCoord1[0]*pixelCoord2[1] - pixelCoord2[0] *
            pixelCoord1[1];
sfn[0][2] = sfn[1][0]*sfn[2][1] - sfn[1][1]*sfn[2][0];
sfn[1][2] = sfn[0][1]*sfn[2][0] - sfn[0][0]*sfn[2][1];
sfn[2][2] = sfn[0][0]*sfn[1][1] - sfn[0][1]*sfn[1][0];
for (int32_t i = 1; i < 3; i++) // normalize
{
  double norm = 0;
  for (int32_t j = 0; j < 3; j++) norm += sfn[j][i]*sfn[j][i];
  norm = sqrt(norm);
  for (int32_t j = 0; j < 3; j++) sfn[j][i] /= norm;
}
double rotationMatrix[3][3] = {0, };
for (int32_t i = 0; i < 3; i++)
{
  for (int32_t j = 0; j < 3; j++)
  {
    for (int32_t k = 0; k < 3; k++) rotationMatrix[i][j] += sfn[i][k]*ifn[j][k];
  }
}
end = steady_clock::now();
cout << "elapsed time: " << duration_cast<milliseconds>(end –
       begin).count() << "ms" << endl;
if (saveCSV)
{
  cout << "saving results in csv format..." << endl;
  csv<double> csvout("7rotation_Matrix.csv");
  vector<string> header;
  vector<vector<double>> fields;
  for (int32_t i = 0; i < 3; i++) header.push_back("C" + to_string(i + 1));
  for (int32_t i = 0; i < 3; i++) fields.push_back(vector<double>(3));
  for (int32_t i = 0; i < 3; i++)
  {
    for (int32_t j = 0; j < 3; j++) fields[i][j] = rotationMatrix[i][j];
  }
  csvout.write(header, fields);
  cout << "saved 7rotation_Matrix.csv" << endl;
}
```

```
  //for debugging purposes
    if (debugMode)
    {
      cout << endl;
      cout << pyramid_starID.size() << " possible pyramid match from catalog
            found" << endl;
      cout << "focal length in pixels: " << f << endl;
      cout << "skyCoord1" << endl;
      for (int32_t i = 0; i < 3; i++) cout << skyCoord1[i] << " ";
      cout << endl;
      cout << "skyCoord2" << endl;
      for (int32_t i = 0; i < 3; i++) cout << skyCoord2[i] << " ";
      cout << endl;
      cout << "pixelCoord1" << endl;
      for (int32_t i = 0; i < 3; i++) cout << pixelCoord1[i] << " ";
      cout << endl;
      cout << "pixelCoord2" << endl;
      for (int32_t i = 0; i < 3; i++) cout << pixelCoord2[i] << " ";
      cout << endl;
      cout << "rotMat" << endl;
      for (int32_t i = 0; i < 3; i++)
      {
        for (int32_t j = 0; j < 3; j++) cout << rotationMatrix[i][j] << " ";
        cout << endl;
      }
      cout << endl;
    }
  }
  void findStar(const char* filename, int32_t sigma, int32_t clusterMinSize,
       int32_t clusterMaxSize, bool saveCSV, bool debugMode)
  {
//1. read image
  cout << "1. read image" << endl;
  steady_clock::time_point begin = steady_clock::now();
  FILE *in = fopen(filename, "rb");
//read header & extract row/column size
  uint8_t header[54];
  fread(header, sizeof(uint8_t), 54, in);
  row = *(int32_t*)(header + 22);
  column = *(int32_t*)(header + 18);
//read raw image
  int32_t size = row*column;
  uint8_t* bgr = new uint8_t[3*size];
  uint8_t* data = new uint8_t[size];
  fread(bgr, sizeof(uint8_t), 3*size, in);
```

```
  for (int32_t i = 0; i < row; i++) // (image stored left to right from bottom row
       to top row)
  {
    for (int32_t j = 0; j < column; j++)
    {
      data[(row - 1 - i)*column + j] = (bgr[3*(i*column + j)] + bgr[3*(i*column
           + j) + 1] + bgr[3*(i*column + j) + 2])/3;
    }
  }
  steady_clock::time_point end = steady_clock::now();
  cout << "elapsed time: " << duration_cast<milliseconds>(end - begin).count()
       << "ms" << endl;
  if (saveCSV)
  {
    cout << "saving results in csv format..." << endl;
    csv<int32_t> csvout("1original_image.csv");
    vector<string> header;
    vector<vector<int32_t>> fields;
    for (int32_t i = 0; i < column; i++) header.push_back("C"+ to_string(i + 1));
    for (int32_t i = 0; i < row; i++)
    {
      fields.push_back(vector<int32_t>());
      for (int32_t j = 0; j < column; j++)fields.back().push_back((int32_t)
          data[i*column + j]);
    }
    csvout.write(header, fields);
    cout << "saved 1original_image.csv" << endl;
  }
//2. thresholding (global)
  cout << "2. thresholding" << endl;
  begin = steady_clock::now();
  int32_t sum = 0;
  int64_t ssum = 0;
  float mean = 0;
  float stddev = 0;
  for (int32_t i = 0; i < size; i++)
  {
    sum += (int32_t)data[i];
    ssum += ((int64_t)data[i])*((int64_t)data[i]);
  }
  mean = (float)sum/size;
  stddev = sqrtf((float)ssum/size - mean*mean);
  for (int32_t i = 0; i < size; i++) if (data[i] < sigma*stddev) data[i] = 0;
  end = steady_clock::now();
  cout << "elapsed time: " << duration_cast<milliseconds>(end - begin).count()
       << "ms" << endl;
```

```
  if (saveCSV)
  {
    cout << "saving results in csv format..." << endl;
    csv<int32_t> csvout("2threshold_image.csv");
    vector<string> header;
    vector<vector<int32_t>> fields;
    for (int32_t i = 0; i < column; i++) header.push_back("C"+ to_string(i + 1));
    for (int32_t i = 0; i < row; i++)
    {
      fields.push_back(vector<int32_t>());
      for (int32_t j = 0; j < column; j++)fields.back().push_back ((int32_t)
            data[i*column + j]);
    }
    csvout.write(header, fields);
    cout << "saved 2threshold_image.csv" << endl;
  }
//3. find and label connected components
  int32_t* nonZeroCluster = new int32_t[size];
  int32_t* nonZeroClusterSize = new int32_t[size];
  int32_t numberofCluster = 0;
  map<int32_t, int32_t> nonZeroCluster_dict;
  map<int32_t, int32_t> nonZeroClusterSize_dict;
  int32_t numberofCluster_dict = 0;
  int32_t connectivity8[8] = { -column - 1, -column, -column + 1, -1, 1,
       column - 1, column, column + 1 };
//crude implementation (with arrays)
  cout << "3. clustering & labelling (with arrays)" << endl;
  begin = steady_clock::now();
//initialize array
  for (int32_t i = 0; i < size; i++)
  {
    nonZeroCluster[i] = -1;
    nonZeroClusterSize[i] = 0;
  }
//find non-zero connected component (excluding single pixel clusters) and
      assign a parent label
  for (int32_t i = 0; i < size; i++)
  {
    if (data[i] > 0) // for non-zero pixels
    {
      int32_t minimumNeighborID = size;
      int32_t myID = i;
      bool isolated = true;
    //find minimum neighbor ID (excluding self)
      for (int32_t j = 0; j < 8; j++)
      {
```

```
//for admissible indices
  if (-1 < i + connectivity8[j] && i + connectivity8[j] < size)
  {
  //for ID assigned non-zero pixels
    if (data[i + connectivity8[j]] > 0)
    {
      if (nonZeroCluster[i + connectivity8[j]] > -1)
      {
        int32_t rootID = nonZeroCluster[i + connectivity8[j]];
        while (rootID != nonZeroCluster[rootID]) rootID =
            nonZeroCluster[rootID]; // find root
        if (minimumNeighborID > rootID) minimumNeighborID = rootID;
      }
      isolated = false;
    }
  }
}
if (!isolated) // exclude single pixel clusters
{
//update self
  if (minimumNeighborID >= myID) minimumNeighborID = i;
    nonZeroCluster[i] = minimumNeighborID;
  nonZeroClusterSize[minimumNeighborID]++;
//update neighbors
  for (int32_t j = 0; j < 8; j++)
  {
    if (-1 < i + connectivity8[j] && i + connectivity8[j] < size) // for
        admissible indices                                        {
    if (data[i + connectivity8[j]] > 0) // for non-zero pixels
    {
      if (nonZeroCluster[i + connectivity8[j]] < 0)//unassinged non-zero pixels
      {
        nonZeroCluster[i + connectivity8[j]] = minimumNeighborID;
      }
      else // ID assigned non-zero pixels
      {
        int32_t rootID = nonZeroCluster[i + connectivity8[j]];
        while (rootID != nonZeroCluster[rootID]) rootID =
            nonZeroCluster[rootID]; // find root
        if (rootID > minimumNeighborID) nonZeroCluster[rootID] =
            minimumNeighborID; // update root
        nonZeroCluster[i+connectivity8[j]]=minimumNeighborID;//update
            neighbor
        }
      }
    }
```

```
        }
       }
      }
     }
    //post processing
     for (int32_t i = 0; i < size; i++)
     {
      if (nonZeroCluster[i] != -1 && nonZeroCluster[i] != i) // for non-zero, non-root,
          non-single pixel cluster pixels
      {
       int32_t rootID = nonZeroCluster[i];
       while (rootID != nonZeroCluster[rootID]) rootID = nonZeroCluster[rootID];
      //find root
       nonZeroCluster[i] = rootID; // update cluster ID
       if (nonZeroClusterSize[i] != 0)
       {
        nonZeroClusterSize[nonZeroCluster[i]] += nonZeroClusterSize[i]; // update
            cluster size
        nonZeroClusterSize[i] = 0;
       }
      }
      if (nonZeroCluster[i] == i) numberofCluster++;
     }
     end = steady_clock::now();
     cout << "elapsed time: " << duration_cast<milliseconds>(end - begin).count() <<
          "ms" << endl;
     cout << "memory usage: " << 2*4*size/1000 << "Kbyte" << endl;
     if (saveCSV)
     {
      cout << "saving results in csv format..." << endl;
      csv<int32_t> csvout("3non-zero_pixel_map.csv");
      vector<string> header;
      vector<vector<int32_t>> fields;
      for (int32_t i = 0; i < column; i++) header.push_back("C" + to_string(i + 1));
      for (int32_t i = 0; i < row; i++)
      {
       fields.push_back(vector<int32_t>());
       for (int32_t j = 0; j < column; j++)    fields.back().push_back
           (nonZeroCluster[i*column + j]);
      }
      csvout.write(header, fields);
      cout << "saved 3non-zero_pixel_map.csv" << endl;
     }
     if (saveCSV)
     {
      cout << "saving results in csv format..." << endl;
```

```
    csv<int32_t> csvout("3non-zero_cluster_size.csv");
    vector<string> header;
    vector<vector<int32_t>> fields;
    for (int32_t i = 0; i < column; i++) header.push_back("C" + to_string(i + 1));
    for (int32_t i = 0; i < row; i++)
    {
      fields.push_back(vector<int32_t>());
      for (int32_t j = 0; j < column; j++) fields.back().push_back
          (nonZeroClusterSize[i*column + j]);
    }
    csvout.write(header, fields);
    cout << "saved 3non-zero_cluster_size.csv" << endl;
  }
//memory optimized implementation (with map STL)
  cout << "3. clustering & labelling (with map STL)" << endl;
  begin = steady_clock::now();
//find non-zero connected component (excluding single pixel clusters) and assign a
parent label
  for (int32_t i = 0; i < size; i++)
  {
    if (data[i] > 0) // for non-zero pixels
    {
      int32_t minimumNeighborID = size;
      int32_t myID = i;
      bool isolated = true;
    //find minimum neighbor ID (excluding self)
      for (int32_t j = 0; j < 8; j++)
      {
      //for admissible indices
        if (-1 < i + connectivity8[j] && i + connectivity8[j] < size)
        {
        /*(little bit slower)
        //for ID assigned non-zero pixels
          if (data[i + connectivity8[j]] > 0)
          {
            if (nonZeroCluster_dict.count(i + connectivity8[j]))
            {
              int32_t rootID = nonZeroCluster_dict[i + connectivity8[j]];
              while (rootID != nonZeroCluster_dict[rootID]) rootID =
                  nonZeroCluster_dict[rootID]; // find root
              if (minimumNeighborID > rootID) minimumNeighborID = rootID;
            }
            isolated = false;
          }
          */
```

```
    if (data[i + connectivity8[j]] > 0)
    {
      if (j < 4) // already registered in nonZeroCluster_dict
      {
        int32_t rootID = nonZeroCluster_dict[i + connectivity8[j]];
        int32_t rootID_ = rootID + 1;
        while (rootID != rootID_) // find root
        {
          rootID_ = rootID;
          rootID = nonZeroCluster_dict[rootID];
        }
        if (minimumNeighborID > rootID) minimumNeighborID = rootID;
      }
      else if (j < 7) // may or may not be registered in nonZeroCluster_dict
      {
        if (nonZeroCluster_dict.find(i + connectivity8[j]) !=
              nonZeroCluster_dict.end())
        {
          int32_t rootID = nonZeroCluster_dict[i + connectivity8[j]];
          int32_t rootID_ = rootID + 1;
          while (rootID != rootID_) // find root
          {
            rootID_ = rootID;
            rootID = nonZeroCluster_dict[rootID];
          }
         if (minimumNeighborID > rootID) minimumNeighborID = rootID;
        }
      }
      else // not registered in nonZeroCluster_dict
      {
      //do nothing
      }
      isolated = false;
    }
  }
}
if (!isolated) // exclude single pixel clusters
{
//update self
  if (minimumNeighborID >= myID) minimumNeighborID = i;
  nonZeroCluster_dict[i] = minimumNeighborID;
  nonZeroClusterSize_dict[minimumNeighborID]++;
//update neighbors
  for (int32_t j = 0; j < 8; j++)
  {
    if (-1<i+connectivity8[j]&&i+connectivity8[j]<size)//for admissible indices
```

```
{
  if (data[i + connectivity8[j]] > 0) // for non-zero pixels
  {
/* (little bit slower)
    if (nonZeroCluster_dict.count(i + connectivity8[j]) == 0) // unassinged
        non-zero pixels
    {
      nonZeroCluster_dict[i + connectivity8[j]] = minimumNeighborID;
    }
    else // ID assigned non-zero pixels
    {
      int32_t rootID = nonZeroCluster_dict[i + connectivity8[j]];
      int32_t rootID_ = rootID + 1;
      while (rootID != rootID_) // find root
      {
        rootID_ = rootID;
        rootID = nonZeroCluster_dict[rootID];
      }
      if (rootID > minimumNeighborID) nonZeroCluster_dict[rootID] =
          minimumNeighborID; // update root
      nonZeroCluster_dict[i + connectivity8[j]] = minimumNeighborID; //
          update neighbor
    }
*/
    if (j < 4) // already registered in noneroCluster_dict
   {
    int32_t rootID = nonZeroCluster_dict[i + connectivity8[j]];
    int32_t rootID_ = rootID + 1;
    while (rootID != rootID_) // find root
    {
      rootID_ = rootID;
      rootID = nonZeroCluster_dict[rootID];
    }
    if (rootID > minimumNeighborID) nonZeroCluster_dict[rootID] =
       minimumNeighborID; // update root
    nonZeroCluster_dict[i + connectivity8[j]] = minimumNeighborID; //
       update neighbor
  }
  else if (j < 7) // may or may not be registered in nonZeroCluster_dict
  {
    if (nonZeroCluster_dict.find(i + connectivity8[j]) ==
       nonZeroCluster_dict.end()) // unassinged non-zero pixels
    {
      nonZeroCluster_dict[i + connectivity8[j]] = minimumNeighborID;
    }
    else // ID assigned non-zero pixels
```

```
                {
                  int32_t rootID = nonZeroCluster_dict[i + connectivity8[j]];
                  int32_t rootID_ = rootID + 1;
                  while (rootID != rootID_) // find root
                  {
                    rootID_ = rootID;
                    rootID = nonZeroCluster_dict[rootID];
                  }
                  if (rootID > minimumNeighborID) nonZeroCluster_dict[rootID] =
                      minimumNeighborID; // update root
                  nonZeroCluster_dict[i + connectivity8[j]] = minimumNeighborID; //
                      update neighbor
                }
              }
              else // not registered in nonZeroCluster_dict
              {
                nonZeroCluster_dict[i + connectivity8[j]] = minimumNeighborID;
              }
            }
          }
        }
      }
    }
  //post processing
    for (auto it = nonZeroCluster_dict.begin(); it != nonZeroCluster_dict.end(); it++)
// for non-zero, non-single pixel cluster pixels
  {
    if (it->first != it->second) // for non-root cluster pixels
    {
      int32_t rootID = it->second;
      while (rootID != nonZeroCluster_dict[rootID]) rootID =
          nonZeroCluster_dict[rootID]; // find root
      it->second = rootID; // update cluster ID
      if (nonZeroClusterSize_dict.find(it->first) != nonZeroClusterSize_dict.end())
      {
        nonZeroClusterSize_dict[it->second] += nonZeroClusterSize_dict[it->first]; //
            update cluster size
        nonZeroClusterSize_dict.erase(it->first);
      }
    }
    else numberofCluster_dict++; // for root cluster pixels
  }
  end = steady_clock::now();
  cout << "elapsed time: " << duration_cast<milliseconds>(end - begin).count() <<
      "ms" << endl;
```

```
  cout << "memory usage: " << (nonZeroCluster_dict.size() +
     nonZeroClusterSize_dict.size())*8/1000 << "Kbyte" << endl;
  if (saveCSV)
  {
    cout << "saving results in csv format..." << endl;
    csv<int32_t> csvout("3non-zero_pixel_map_STL.csv");
    vector<string> header;
    vector<vector<int32_t>> fields;
    for (int32_t i = 0; i < column; i++) header.push_back("C" + to_string(i + 1));
    for (int32_t i = 0; i < row; i++) fields.push_back(vector<int32_t>(column, -1));
  //initialize with -1
    for (auto it = nonZeroCluster_dict.begin(); it != nonZeroCluster_dict.end(); it++)
    {
      fields[(it->first)/column][(it->first)%column] = it->second;
    }
    csvout.write(header, fields);
    cout << "saved 3non-zero_pixel_map_STL.csv" << endl;
  }
  if (saveCSV)
  {
    cout << "saving results in csv format..." << endl;
    csv<int32_t> csvout("3non-zero_cluster_size_STL.csv");
    vector<string> header;
    vector<vector<int32_t>> fields;
    for (int32_t i = 0; i < column; i++) header.push_back("C" + to_string(i + 1));
    for (int32_t i = 0; i < row; i++) fields.push_back(vector<int32_t>(column, 0));
// initialize with 0
    for (auto it = nonZeroClusterSize_dict.begin(); it !=
       nonZeroClusterSize_dict.end(); it++)
    {
      fields[(it->first)/column][(it->first)%column] = it->second;
    }
    csvout.write(header, fields);
    cout << "saved 3non-zero_cluster_size_STL.csv" << endl;
  }
//4. find centroid
  cout << "4. get cendtroid" << endl;
  begin = steady_clock::now();
  vector<float> centroid_pixelx;
  vector<float> centroid_pixely;
//array for clustering
/*for (int32_t i = 0; i < size; i++)
  {
    if (nonZeroClusterSize[i] > clusterMinSize && nonZeroClusterSize[i] <
        clusterMaxSize)
    {
```

```
      int32_t window = i + connectivity8[7] + 1;
      int32_t sumx = 0;
      int32_t sumy = 0;
      int32_t sum = 0;
      for (int32_t j = i; j < window && j < size; j++)
      {
       if (nonZeroCluster[j] == i)
       {
        sumx += (j%column)*data[j];
        sumy += (j/column)*data[j];
        sum += data[j];
        window = j + connectivity8[7] + 1;
       }
      }
      centroid_x.push_back(((float)sumx)/sum + 0.5f - 0.5f*column);
      centroid_y.push_back(0.5f*row - ((float)sumy)/sum - 0.5f);
      centroid_pixelx.push_back(((float)sumx)/sum);
      centroid_pixely.push_back(((float)sumy)/sum);
     }
    }*/
  //map for clustering
    for (auto it = nonZeroClusterSize_dict.begin(); it !=
         nonZeroClusterSize_dict.end(); it++)
    {
     if (it->second > clusterMinSize && it->second < clusterMaxSize)
     {
      int32_t window = it->first + connectivity8[7] + 1;
      int32_t sumx = 0;
      int32_t sumy = 0;
      int32_t sum = 0;
      int32_t count = 0;
      for (auto it2 = nonZeroCluster_dict.find(it->first); it2 !=
           nonZeroCluster_dict.end() && count < it->second; it2++)
      {
       if (it2->second == it->first)
       {
        int32_t index = it2->first;
        sumx += (index%column)*data[index];
        sumy += (index/column)*data[index];
        sum += data[index];
        count++;
       }
      }
      centroid_x.push_back(((float)sumx)/sum + 0.5f - 0.5f*column);
      centroid_y.push_back(0.5f*row - ((float)sumy)/sum - 0.5f);
      centroid_pixelx.push_back(((float)sumx)/sum);
```

```
      centroid_pixely.push_back(((float)sumy)/sum);
    }
   }
   end = steady_clock::now();
   cout << "elapsed time: " << duration_cast<milliseconds>(end - begin).count()
        << "ms" << endl;
   if (saveCSV)
   {
    cout << "saving results in csv format..." << endl;
    csv<float> csvout("4centroid.csv");
    vector<string> header;
    vector<vector<float>> fields;
    header.push_back("myID"); header.push_back("X"); header.push_back("Y");
    header.push_back("X'"); header.push_back("Y'");
    for (uint32_t i = 0; i < centroid_x.size(); i++)
    {
     fields.push_back(vector<float>());
     fields.back().push_back(i);
     fields.back().push_back(centroid_x[i]); fields.back().push_back(centroid_y[i]);
     fields.back().push_back(centroid_pixelx[i]);
     fields.back().push_back(centroid_pixely[i]);
    }
    csvout.write(header, fields);
    cout << "saved 4centroid.csv" << endl;
   }
  //for debugging purposes
   if (debugMode)
   {
    cout << endl;
    cout << "width(columns): " << column << ", height(rows): " << row << endl;
    cout << "sum: " << sum << ", ssum: " << ssum << endl;
    cout << "mean: " << mean << ", stddev: " << stddev << endl;
    cout << "number of non-zero clusters: " << numberofCluster << endl;
    cout << "number of stars: " << centroid_x.size() << endl;
    cout << endl;
   }
  //return
   fclose(in);
   delete[] bgr;
   delete[] data;
   delete[] nonZeroCluster;
   delete[] nonZeroClusterSize;
  }
```

Appendix C: List of Star Constellations Tested in Simulator

Format: [star constellation, Four HIP ID, (Right Ascension, Declination, Rotation)]

1. CASSIOPEIA: 4422/4292/3821/4427 (14.17, 60.71,180)
2. LYRA: 93017,93194,93279,93917 (281.1933, 31.6052,0)
3. ORION: 25473,25302,25142,25861 (82,8,90)
4. Delphinus: 101958,102281, 101769, 101589 (310,14,90)
5. Perseus:16335,15770,16147,16244 (60,40,0)
6. Columba: 27628,26868,28199,27810 (90,-36,0)
7. Acquarius: 114939,114855,115033,115115 (348.75,-10,180);
8. Andromeda:6813,6411,6999,5434(15,41,0)
9. Aquila:97278,97675,96957,97649 (292.5,8,45);
10. Aries: 9836,9884,9153,10053(37.5,20,0);
11. Auriga: 28380,27639,28946,27673(90,45,0);
12. Bootes:67927,67275,67459,67480 (210,18,90)
13. Camelopardalis:21727, 23040, 19949, 22287 (75,60,)
14. Cancer:42911,42806, 43970, 43834 (127.5,20, 180)
15. Canes venatici: 63901, 63125, 64540, 64844 (195, 40 ,0)
16. Capricornus: 104963, 105143, 104019, 104365 (318,-20,0)
17. Centaurus: 68245, 67464, 68862, 67472 (210,-34, 0)
18. Cepheus: 109400, 108535, 107230, 11532 (330, 74, 0)
19. Cetus: 11738, 12706, 12093, 12387 (45, 6, 0)
20. Chameleon: 58484,60000, 52633, 58905 (165, -80, 180)
21. Circinus: 74778, 74941, 75323, 74824 (229.5, -60, 0)
22. Comaberenices: 60514, 60351, 60746, 60904 (187.5, 24, 0)
23. Corona Australis: 94114,93825,93174, 94005 (289.5, -40, 0)
24. Corona Borealis: 78159, 78493, 77512, 76952 (243, 34, 0)
25. Corvus: 60965, 59803, 61174, 60221 (187.5, -16, 0)
26. Crater: 55598, 55705, 55282,57283 (169.5, -14, 0)
27. Crux: 62434, 61966, 63007, 62268 (192, -64, 0)
28. Cygnus: 103089,102724, 103732,102843 (315,46,0)
29. Dorado: 29353, 27100, 27890, 27534 (82.5, -64, 0)
30. Draco: 85670, 85819, 85829, 83608 (262.5, 56, 0)
31. Equuleus: 105570, 104858, 104521, 104987 (318, 7, 0)
32. Eridanus: 17593,18543,19777,17378 (60,-10,0)
33. Fornax: 11918, 13202,11072,10320 (37.5, -30 ,0)
34. Gemini:33202,32249,32362,34033 (105, 20,0)
35. Grus: 112122, 111043, 110997, 114421 (345, -50,0)
36. Hercules: 88886,88331,88657,88267 (270,20,0)
37. Hyrdra: 52085,52943,51069,52009 (157.5, -16,0)
38. Hydrus: 11001,12394,12876,8928 (30, -70,0)
39. Lacerta:112242,112031,111944,111104 (337.5,40,90)
40. Leo: 49669,48883,49583,49637 (150,20,0)
41. Leo minor: 53229,52098,52638,51056(154.5,32,90)
42. Lepus: 24845,24873,24327,24244 (82.5,-20,0)

43. Libra:76219,76243,74785,75379 (229.5,-10,0)
44. Lupus:69996,70574,74117,70104(225,-46,0)
45. Lynx:37701,35384,39847,33449 (118.5, 50,0)
46. Microscopium:103882,102831,104174,104738 (315,-36,0)
47. Monoceros:31978,32533,31216,31119 (99,5,180)
48. Musca:56862,57851,5597,57363 (187.5, -75,0)
49. Norma:79790,80208,78639,80000 (243,-50,0)
50. Ophiuchus:88290,88601,88192,88149 (264,5,0)
51. Pave: 101612,103227,100751,101983 (300,-64,0)
52. Pegasus: 114155,115250,112748 (345,20,180)
53. Phoneix: 3245,2072,2081,765 (7.5, -45, 0)
54. Pictor: 29276,27530,27947,27621 (82.5,-50,0)
55. Piscses: 4906, 5737, 3786,6061 (15,10,90)
56. Piscis Austrinus: 108661, 109422, 109789, 109285 (330,-30,0)
57. Pupis:38020,38164,37606,38089 (106.5,-45,0)
58. Pyxis:42828, 43352, 42483, 42515 (135,-30,0)
59. Reticulum: 19780,18744,18772,18597 (60,-60,0)
60. Sagitta:97365,97496,96757,96837 (300,20,0)
61. Sagittarrus: 93085,93057,92845,93683 (285, -30, 0)
62. Scorpius;(255,-35)
63. Sculptor: 183, 117452,1708,2381 (0,-32,0)
64. Scutum: 91105, 91726, 91117, 91845 (280.5,10,0)
65. Serpens: 76425,76276,76866,77257 (235.5,6)
66. Sextans: (153,0,0)
67. Taurus:19740,19719,19860,19799 (60,6,0)
68. Telescopium:93148,93815,95261,92646 (292.5,-50,180)
69. Triangulum:10559,10670,10644,10064 (30,32,0)
70. Triangulum Australae: 75323,74824,74941,74778 (240,-64,90)
71. Tucana:111310,110838,110130,114996 (350,-64,0)
72. Ursa Major:53295,54539,52469,53721, (165,50,0)
73. Ursa Minor: 72607, 70692, 76008, 69112 (225,75,0)
74. Vela: 45505,45344,45085,44816 (150,-50,90)
75. Virgo: 65474,64078,65581,66098 (202.5,10,90)
76. Volan: 44626,44599,42425,45328 (120,-68,90)
77. Vulpecula: 96757,97365,9627,96837 (300,24,0)